Models, constructive heuristics, and benchmark instances for the flexible job shop scheduling problem with nonlinear routes and position-based learning effect*

K. A. G. Araujo† E. G. Birgin‡ D. P. Ronconi§

April 1, 2024

Abstract

This paper addresses the flexible job shop scheduling problem with nonlinear routes and position-based learning effect. In this variant of the flexible job shop scheduling problem, precedence constraints of the operations constituting a job are given by an arbitrary directed acyclic graph, generalizing the classical case in which a total order is imposed. Additionally, it is assumed that the processing time of an operation in a machine is subject to a learning process such that the larger the position of the operation in the machine, the faster the operation is processed. Mixed integer programming and constraint programming models are presented and compared in the present work. In addition, constructive heuristics are introduced to provide an initial solution to the models' solvers. Sets of benchmark instances are also introduced. The problem considered corresponds to modern problems of great relevance in the printing industry. The models and instances presented are intended to support the development of new heuristic and metaheuristics methods for this problem.

Keywords: flexible job shop scheduling problem, nonlinear routes, learning effect, models, instances, constructive heuristics.

1 Introduction

In this work, we consider the flexible job shop (FJS) scheduling problem with nonlinear routes and position-based learning effect. The problem is NP-hard, as it has the job shop scheduling problem, known to be NP-hard [13], as a particular case. The nonlinear routes feature refers

*This work has been partially supported by the Brazilian agencies FAPESP (grants 2013/07375-0, 2022/05803-3, and 2023/08706-1) and CNPq (grants 311536/2020-4 and 302073/2022-1).

†Department of Applied Mathematics, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil. e-mail: kennedy94@ime.usp.br

‡Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil. e-mail: egbirgin@ime.usp.br. Corresponding author.

§Department of Production Engineering, Polytechnic School, University of São Paulo, Av. Prof. Luciano Gualberto, 1380, Cidade Universitária, 05508-010 São Paulo, SP, Brazil. e-mail: dronconi@usp.br

to the fact that the precedence constraints imposed on the operations of a job are given by an arbitrary directed acyclic graph instead of the usual linear order enforced in the FJS scheduling. Many real-life scheduling problems fit into this scope, such as, for example, problems in the printing industry [4, 21, 22], mold manufacturing industry [11], and glass industry [1]. In [22], the FJS with nonlinear routes and several additional features, such as, for example, resumable operations, periods of unavailability of the machines, sequence-dependent setup times, partial overlapping between operations with precedence constraints, and fixed operations, was addressed. However, learning effects have not yet been taken into account in the literature regarding FJS with nonlinear routes. The present work is devoted to this problem and presents the first step towards its effective and efficient resolution. Mixed integer linear programming (MILP) and constraint programming (CP) models and a relatively large set of instances are introduced for the purpose of producing a benchmark test set. Since, leaving aside very small instances, commercial exact solvers alone are hardly even able to find a feasible solution, constructive heuristics are proposed with the goal of enhancing their performance. Overall, this work takes a first step towards solving the proposed problem and provides a solid and robust basis for the future development of more sophisticated heuristic and metaheuristic methods.

In classical scheduling problems, the processing time of a given operation on a given machine is a fixed input parameter. However, there are real-life situations in which the manufacturing process involves repetitive manual tasks and the worker undergoes a learning process that results in a reduction of the execution time of his/her task. For instance, workers can get proficient at performing assemblies quickly, or more confident and skillful in manipulating hardware, software, and/or raw materials. If we consider that the reduction in the operation processing time is related to the number of times the worker has already performed the operation, we are dealing with a position-based learning effect. The pioneering works in applying the concept of learning effect to scheduling problems are [6, 9, 14]. Surveys on the subject can be found in [3, 7, 15].

A brief literature review in chronological order of the FJS with nonlinear routes follows. In [12], the problem was addressed by considering non-fixed intervals of machine unavailability for preventive maintenance. A multi-objective MILP formulation and a hybrid multi-objective genetic algorithm were proposed. In [11] and [18], process flexibility was also taken into account, which means that the same result can be obtained with different operation sequences. In [11] a branch-and-bound algorithm was proposed, while [18] considered a symbiotic evolutionary algorithm. In [24], where a MILP formulation was presented to minimize the weighted tardiness, a hybrid bacterial foraging optimization algorithm was developed. Furthermore, the algorithm was enhanced by a local search method based on the manipulation of critical operations. A research that addresses the FJS with nonlinear routes and sequence-dependent setup times can be found in [8]. The authors proposed a knowledge-based cuckoo search algorithm associated with a reinforcement learning strategy for self-adjustment. In [17], MILP and CP models and a hybrid evolutionary algorithm with local search mechanisms were introduced. A variation in which the processing of each operation requires multiple resources was considered in [16], in which models are presented and some properties of the problems are analyzed.

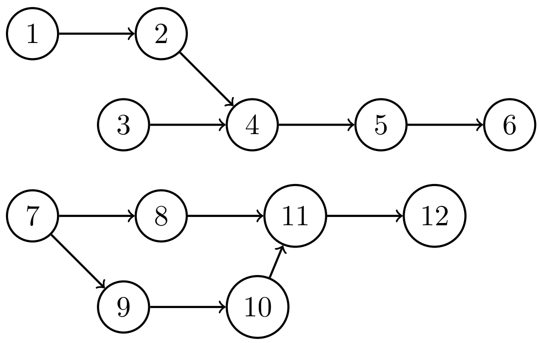
In [1], an application in the glass industry was described and a heuristic approach combining local search and priority rules was proposed to minimize the total cost related to final product completion times. Other industrial environments, such as the printing industry, have also been modeled as an FJS with nonlinear routes. Regarding this particular application, [23] suggested a bi-objective genetic algorithm based on NSGA II to solve the problem. In [4], a MILP model and a constructive heuristic were presented, while [5] introduced a list scheduling algorithm and its extension to a beam search method. In [21, 22], formulations using constrained programming and mixed integer linear programming were established, as well as trajectory and population metaheuristics were introduced. In [27], it was considered the flight deck scheduling problem, which is an FJS with nonlinear routes with additional constraints that state that some operations must be completed before others. The problem was described through its MILP formulation and instances were solved with a differential evolution type method. In [2], the scheduling of repair orders and worker assignment in an automotive repair shop was analyzed. The main scheduling problem is a two-resource FJS with nonlinear routes. The problem was modeled by extending the formulation introduced in [4] and an iterated greedy heuristic was presented.

Let us consider an illustrative example of the FJS with nonlinear routes and position-based learning effect. The example has 12 operations and 3 machines. Figure 1 shows the precedence relationships between the operations and the standard processing times of each operation on each machine. The small table with the standard processing times shows that in the FJS scheduling problem each operation can be processed by one or more machines (situation known as routing flexibility), generalizing the job shop (JS) scheduling problem in which each operation can be processed by only one machine. The concept of job is implicitly defined by the directed acyclic graph (DAG) and corresponds to a set of operations that have some dependency relationship between them. The figure makes it clear that, in the FJS scheduling problem with nonlinear routes, the dependencies of the operations of a job are given by an arbitrary acyclic directed graph instead of the total order of the FJS scheduling problem. In this example there are two jobs, one with 6 operations (numbered from 1 to 6) and the other also with 6 operations (numbered from 7 to 12).

A feasible solution to the instance of Figure 1 can be illustrated by a DAG in which a source node s and a target node t are added to the DAG that represents the precedence constraints; see Figure 2. Arcs from s to all operations with no predecessors and from all nodes without successor to t must also be added. (For the readers that can see the figure in colors, those arcs are painted purple.) In addition, dashed arcs represent the sequence (list) in which operations are processed by each machine. (For the readers that can see the figure in colors, blue corresponds to machine 1, violet corresponds to machine 2, and orange corresponds to machine 3.) Colored figures at the top or bottom of the operations correspond to their processing times. In this directed graph, that we name $G = (V, A)$ from now on[1], the longest path from s to t corresponds to the makespan. In this example, the longest path, with value 80, corresponds to the path $s, 1, 2, 4, 5, 6, t$. (Highlighted yellow in the figure for those readers who can see the figure in color.) The depicted feasible solution corresponds to an optimal solution. Figure 3 shows the Gantt chart representation of the solution. Note that standard times were used, i.e. in this example the learning effect was not considered at all.

In the present work, we consider that when an operation i is assigned to a machine k and it is the rth operation to be processed by the machine, the standard processing time p_{ik} is affected by a learning effect and becomes $\psi_\alpha(p_{ik}, r)$, where $\alpha > 0$ is the learning rate. Following [6],

[1]Note that $V = \mathcal{O} \cup \{s, t\}$ and A is composed by the given arcs in $\widehat{A}$ that represent the precedence relations among operations, plus the mentioned arcs related to the new nodes s and t, plus the mentioned machine arcs.



Operations	Machines 1	Machines 2	Machines 3
1	10	20	15
2	20	15	5
3	10	–	20
4	–	30	–
5	30	40	10
6	20	–	30
7	–	10	20
8	40	10	–
9	–	40	20
10	10	20	10
11	20	10	–
12	–	–	15

Figure 1: On the left, representation of operations' precedence constraints by a directed acyclic graph $D = (\mathcal{O}, \widehat{A})$, where $\mathcal{O} = \{1, 2, \dots, 12\}$ represents the set of operations and $\widehat{A}$ is the set of arcs that represent the precedence constraints. On the right, standard processing times of the twelve operations on each of the three machines. In the table cells, "–" means that the machine cannot process the operation.

we might consider $\psi_\alpha(p, r) := p/r^\alpha$ in which, the larger the value of α, the faster the learning. However, in the present work, a constraint programming model of the problem will be introduced and instances solved with a commercial solver that requires processing times to assume integer values only. Thus, we consider $\psi_\alpha(p, r) := \lfloor 100\, p/r^\alpha + 1/2 \rfloor$. Multiplying by one hundred, adding 0.5, and taking the floor corresponds to changing the unit of measures (from seconds to milliseconds, for example) and rounding to the closest integer value. Figures 4 and 5 illustrate the graph representation and the Gantt chart of an optimal solution in which the learning function ψ_α with $\alpha = 0.5$ is considered. It is worth noting that a different schedule is found, whose makespan, given by the critical path $s, 7, 8, 4, 5, 6, t$, is equal to 50.16 in the original units of time (5016 in the new one).

The rest of this paper is organized as follows. In Section 2, we introduce the integer linear programming and constraint programming models. In Section 3, we outline the proposed constructive heuristics. In Section 4, we report the introduced instances. Numerical experiments with the constructive heuristics and exact commercial solvers are reported in Section 5. Conclusions and lines of future research are presented in the concluding section.

2 Mixed integer and constraint programming models

In this section, we present mixed-integer linear programming (MILP) and constraint programming (CP) formulations for the FJS scheduling problem with nonlinear routes and position-based learning effect.

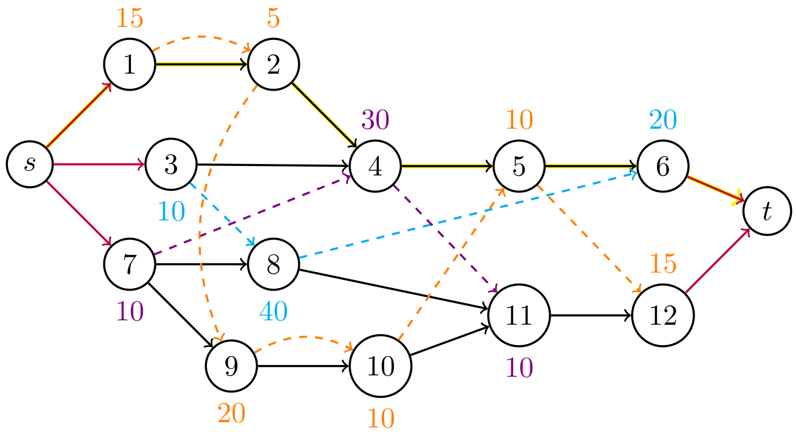


Figure 2: Representation of an optimal solution to the instance in Figure 1.

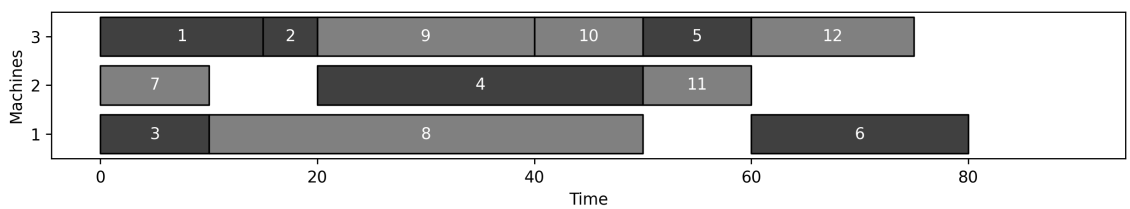


Figure 3: Gantt chart representation of the optimal solution shown in Figure 2 to the instance of Figure 1.

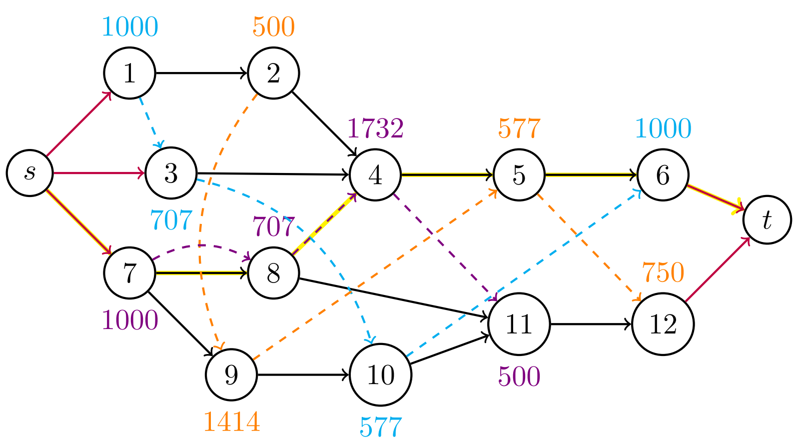


Figure 4: Representation of an optimal solution to the instance in Figure 1 in the presence of learning effect.

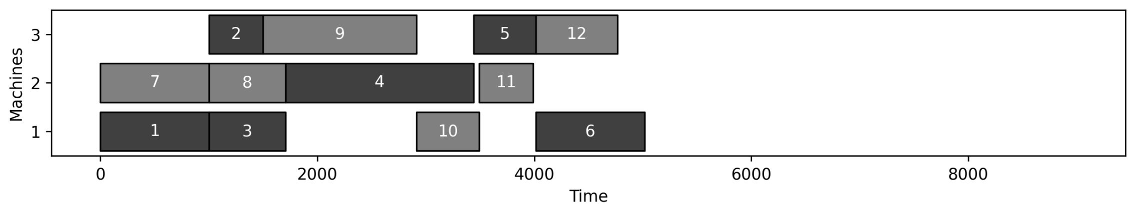


Figure 5: Gantt chart representation of the optimal solution shown in Figure 4 to the instance of Figure 1 in the presence of learning effect.

2.1 Mixed-integer linear programming model

The adoption of position-based decision variables serves as the fundamental approach for modeling problems involving position-based learning effects, as it enables a more natural expression of constraints related to the change in processing times. The proposed MILP model is derived from [4] and is built upon the model introduced in [25] that considers position-based decision variables; see also [26]. Notation below simplifies the presentation of the model.

Sets:

$\mathcal{O}$**:** set of operations,

$\mathcal{F}$**:** set of machines,

$\mathcal{O}_k$**:** set of operations that can be processed by machine k,

$\mathcal{F}_i$**:** set of machines that can process operation i,

$\widehat{A}$**:** set of directed arcs that represent operations' precedence constraints.

Parameters:

p_{ik}**:** standard processing time of operation i in machine k,

$\psi_\alpha(p_{ik}, r)$**:** position-based learning function.

Decision variables:

x_{ikr}**:** 1 if operation i is the rth operation to be processed in machine k; 0, otherwise,

s_i**:** starting time of operation i,

h_{kr}**:** starting time of the rth operation to be processed in machine k,

p'_i**:** actual processing time of operation i (considering the learning effect).

The MILP model follows.

$$\text{Minimize} \quad C_{\max} \tag{1}$$

subject to

$$\sum_{k \in \mathcal{F}_i} \sum_{r=1}^{|\mathcal{O}_k|} x_{ikr} = 1, \quad i \in \mathcal{O}, \tag{2}$$

$$\sum_{i \in \mathcal{O}_k} x_{ikr} \le 1, \quad k \in \mathcal{F}, r = 1, \dots, |\mathcal{O}_k|, \tag{3}$$

$$\sum_{i \in \mathcal{O}_k} x_{i,k,r+1} \le \sum_{i \in \mathcal{O}_k} x_{ikr}, \quad k \in \mathcal{F}, r = 1, \dots, |\mathcal{O}_k| - 1, \tag{4}$$

$$p'_i = \sum_{k \in \mathcal{F}_i} \sum_{r=1}^{|\mathcal{O}_k|} \psi_\alpha(p_{ik}, r)\, x_{ikr}, \quad i \in \mathcal{O}, \tag{5}$$

$$h_{kr} + \sum_{i \in \mathcal{O}_k} \psi_\alpha(p_{ik}, r)\, x_{ikr} \le h_{k,r+1}, \quad k \in \mathcal{F}, r = 1, \dots, |\mathcal{O}_k| - 1, \tag{6}$$

$$h_{kr} + \sum_{i \in \mathcal{O}_k} \psi_\alpha(p_{ik}, r)\, x_{ikr} \le C_{\max}, \quad k \in \mathcal{F}, r = |\mathcal{O}_k|, \tag{7}$$

$$s_i + p'_i \le s_j, \quad \forall (i,j) \in \widehat{A}, \tag{8}$$

$$s_i + p'_i - \left(2 - x_{ikr} - \sum_{t=r+1}^{|\mathcal{O}_k|} x_{jkt}\right) M \le s_j, \quad \begin{array}{c} \forall i,j \in \{\mathcal{O} \mid i \neq j\}, \forall k \in \mathcal{F}_i \cap \mathcal{F}_j, \\ r = 1, \dots, |\mathcal{O}_k| - 1, \end{array} \tag{9}$$

$$h_{kr} - M\,(1 - x_{ikr}) \le s_i, \quad i \in \mathcal{O}, k \in \mathcal{F}_i, r = 1, \dots, |\mathcal{O}_k|, \tag{10}$$

$$s_i - M\,(1 - x_{ikr}) \le h_{kr}, \quad i \in \mathcal{O}, k \in \mathcal{F}_i, r = 1, \dots, |\mathcal{O}_k|, \tag{11}$$

$$s_i \ge 0, \quad i \in \mathcal{O}, \tag{12}$$

$$h_{kr} \ge 0, \quad k \in \mathcal{F}_i, r = 1, \dots, |\mathcal{O}_k|, \tag{13}$$

$$x_{ikr} \in \{0,1\}, \quad i \in \mathcal{O}, k \in \mathcal{F}_i, r = 1, \dots, |\mathcal{O}_k|. \tag{14}$$

Objective function (1) represents the minimization of the makespan. Constraints (2) state that each operation must be processed by exactly one machine and occupy exactly one position on that machine. Constraints (3) state that each position of each machine can be associated with at most one operation. Constraints (4) that a machine position (other than the first position) can be occupied by an operation only if the previous position is also occupied by another operation. Constraints (5) define the actual processing time of each operation i in order to simplify the presentation of the model. Constraints (6) avoid the overlapping of operations assigned to the same machine. Constraints (7) say that the makespan must be greater than or equal to the completion time of all operations. Combining these constraints with the minimization of the objective function (1) makes the makespan to coincide with the completion time of the last operation to be completed. Constraints (8) impose the given precedence constraints between operations by saying that if an operation i precedes an operation j then j cannot be started before i is completed. Constraints (9) state that, if two operations i and j were assigned to the same machine k and operation i precedes operation j, then j cannot be started before i is completed. If operation i is assigned to position r of machine k (i.e. if $x_{ikr} = 1$) then constraints (10) and (11) force h_{kr} (the starting time of rth operation to be processed by machine k) to be equal to s_i (the starting time of operation i). Constraints (12), (13), and (14) determine the decision variables' domains. M is a "sufficiently large" number and may assume the value $\sum_{i\in\mathcal{O}}\sum_{k\in\mathcal{F}} p_{ik}$. Function $\psi_\alpha(p_{ik}, r)$ is the given function that represents the learning effect and computes the actual processing time of operation i if it is assigned to position r of machine k. This function has the learning rate $\alpha \geq 0$ as a parameter and, in the present work, as already mentioned in the introduction, is given by

$$\psi_\alpha(p, r) = \left\lfloor 100\, r^{-\alpha}\, p + \frac{1}{2} \right\rfloor.$$

2.2 Constraint Programming model

Constraint Programming (CP) is a potent methodology widely employed for solving scheduling problems in academic and industrial literature. CP Optimizer [19], being an optimization commercial solver rooted in CP, incorporates specialized concepts of constraints and variables, significantly facilitating the modeling process for scheduling problems. In this section, we introduce a CP model specifically designed for its utilization in connection with CP Optimizer. The syntax of CP Optimizer is defined as it arises within the formulation. The model follows below.

$$\text{Minimize} \quad \max_{i\in\mathcal{O}} \text{endOf}(o_i) \tag{15}$$

$$\text{subject to}$$

$$\text{endBeforeStart}(o_i, o_j), \quad (i,j) \in \widehat{A}, \tag{16}$$

$$\text{alternative}\left(o_i, [a_{ikr}]_{k\in\mathcal{F}_i, r=1,\dots,|\mathcal{O}_k|}\right), \quad i \in \mathcal{O}, \tag{17}$$

$$\text{noOverlap}\left([a_{ikr}]_{i\in\mathcal{O}_k, r=1,\dots,|\mathcal{O}_k|}\right), \quad k \in \mathcal{F}, \tag{18}$$

$$\text{endBeforeStart}(a_{ikr}, a_{jk,r+1}), \quad \begin{array}{c} i,j \in \mathcal{O},\, k \in \mathcal{F}_i \cap \mathcal{F}_j, \\ r = 1,\dots,|\mathcal{O}_k|-1, \end{array} \tag{19}$$

$$\text{or}([\text{presenceOf}(a_{ik,r+1})]_{i\in\mathcal{O}_k}) \implies \text{or}([\text{presenceOf}(a_{ikr}])_{i\in\mathcal{O}_k}), \quad k \in \mathcal{F}, r = 1,\dots,|\mathcal{O}_k|-1, \tag{20}$$

$$\text{interval } o_i, \quad i \in \mathcal{O}, \tag{21}$$

$$\text{interval } a_{ikr},\ \text{opt},\ \text{size } = \psi_\alpha(p_{ik}, r), \quad i \in \mathcal{O}, k \in \mathcal{F}_i, r = 1,\dots,|\mathcal{O}_k|. \tag{22}$$

Interval decision variables of the problem are described in (21) and (22). In (21), an interval variable o_i for each operation i is defined. In (22), an *optional* interval variable a_{ikr} is defined for each possible assignment of operation i to a machine $k \in \mathcal{F}_i$ at positions $r = 1,\dots,|\mathcal{O}_k|$. *Optional* means that the interval variable may exist or not; and the remaining of the constraint says that, in case it exists, its size must be given by $\psi_\alpha(p_{ik}, r)$. Constraints (17) state that each operation i must be allocated to exactly one machine $k \in \mathcal{F}_i$ in exactly one position r, that is, one and only one interval variable a_{ikr} must be present and the selected interval a_{ikr} must start and end at the same instants as interval o_i. The objective function (15) is to minimize the makespan, given by the maximum end value of all the operations represented by the interval variables o_i. Precedence constraints between operations are posted as endBeforeStart constraints between interval variables in constraints (16). Constraints (18) state that, for each machine k, at most one operation can be assigned to each position and operations assigned to different positions cannot overlap. Constraints (19) say that, for any machine and any position, the operation assigned to that position must finish before the operation in the next position starts. Constraints (20) force that the empty positions of machine k are the last ones, i.e., that operations are processed in the first positions of the machines without empty positions among them.

3 Constructive Heuristics

In this section, we propose two constructive heuristics for the FJS scheduling problem with nonlinear routes and position-based learning effect. Constructive heuristics are algorithms that build a feasible solution from scratch by iteratively selecting and sequencing one operation at a time. The two proposed constructive heuristics are based on the earliest starting time (EST) rule [4] and the earliest completion time (ECT) rule [20]. The goal is to use them to provide an initial feasible solution to the exact solver that will be used to attempt to solve a set of test instances.

Algorithm 1 presents the constructive heuristic based on EST rule. In the algorithm, r_v^{op} refers to the ready time of operation v, w_v refers to its actual processing time and c_v to its completion time. On the side of the machines, r_k^{mac} represents the instant in which machine k is released and g_k represents its first free position, which is the one that would be occupied if an operation were assigned to it (both quantities refer to the partial scheduling being constructed). f_v will indicate to which machine operation v was assigned and each machine k will have an ordered list Q_k with the sequence of operations to be processed. After the initializations (lines 2 to 5) comes the main loop, which is executed as long as there are still unscheduled operations. Among the not scheduled ones, the ready time is calculated for all those that already have all the preceding operations scheduled (lines 7 to 9). In line 10, observing the ready times of the operations and machines, the smallest instant $r_{\min}$ in which an operation could be scheduled is calculated and the set E of operation/machine pairs that could start at that instant $r_{\min}$ is constructed. As it was observed in [4], $|E|$ can be quite large and experience shows that a tie-breaking rule can significantly improve the method's performance. Thus, in line 11, among all the operation/machine pairs in E, taking into account the learning effect, the pair $(\hat{v}, \hat{k})$ with the shortest processing time is chosen. In line 12, $w_{\hat{v}}$, $f_{\hat{v}}$, and $c_{\hat{v}}$ are defined and the ready time $r_{\hat{k}}^{\text{mac}}$ and the free position $g_{\hat{k}}$ of machine $\hat{k}$ are updated. In lines 13 and 14 the corresponding machine arc is inserted in the graph G (the arc must not be inserted if operation $\hat{v}$ is the first one of machine $\hat{k}$). Finally, the list of operations assigned to machine $\hat{k}$ is updated and the scheduled operation is removed from the set of operations not yet scheduled. After all the operations have been scheduled, the critical path in G is calculated (line 16) to determine the makespan value $C_{\max}$. This is done with Algorithm 2. Initializations in lines 2 to 5 of Algorithm 1 have complexity $O(|\mathcal{O}| + |\widehat{\mathcal{A}}| + |\mathcal{F}|)$. Within the main loop (lines 6 to 15), lines 7–9 have complexity $O(|\mathcal{A}|) = O(|\widehat{A}|+|\mathcal{O}|+|\mathcal{F}|)$, lines 10–11 have complexity $O(|\mathcal{O}|+\sum_{i \in \mathcal{O}} |F_i|)$, and line 12 has complexity $O(|\mathcal{O}|)$. Since the main loop is executed $|\mathcal{O}|$ times, its total complexity is $O(|\mathcal{O}|(|\widehat{A}|+|\mathcal{F}|+\sum_{i\in\mathcal{O}}|\mathcal{F}_i|))$. As the complexity of the main loop is larger that the complexity of the initialization as well as the complexity of line 16 (see below), then the complexity of Algorithm 1 is given by the complexity of its main loop. It is important to note that $\gamma = \sum_{i\in\mathcal{O}}|\mathcal{F}_i|$ is between $|\mathcal{O}|$ and $|\mathcal{O}||\mathcal{F}|$, but we prefer to keep the complexity expressed as a function of γ because γ is a measure of the size of the input that depends on the routes' nonlinearity of the instance under consideration. It is also important to note that the complexity of the algorithm depends on the routing flexibility of the operations, i.e., it depends on the number of dependency relations in $\widehat{A}$. Therefore, it is important to represent an instance in such a way that $\widehat{A}$ corresponds to a transitive reduction of the precedences' digraph.

Algorithm 1: Computes a solution graph $G=(V,A)$, f, Q, and w by using EST dispatching rule. Then, in G, it computes the largest path $\mathcal{P}$ from s to t and its length $C_{\max}$.

Input: $\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$
Output: f, w, Q, $G=(V,A)$, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ
1 **function** EST($\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$, f, w, Q, G, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ)
2   Set $A \leftarrow \widehat{A} \cup \{(s,j) \mid (\cdot,j) \notin \widehat{A}\} \cup \{(i,t) \mid (i,\cdot) \notin \widehat{A}\}$ and define $V := \mathcal{O} \cup \{s,t\}$ and $G=(V,A)$.
3   Set $r^{\text{op}}_v \leftarrow +\infty$ for all $v \in V$ and define $r^{\text{op}}_s := 0$, $w_s := w_t := 0$, and $c_s := 0$.
4   Set $r^{\text{mac}}_k \leftarrow 0$ and $g_k \leftarrow 1$ for all $k \in \mathcal{F}$.
5   Initialize $\Pi \leftarrow V \setminus \{s,t\}$ as the set of non-scheduled operations, and Q_k as an empty list for all $k \in \mathcal{F}$.
6   **while** $\Pi \neq \emptyset$ **do**
7   **for** $v \in \Pi$ **do**
8   **if** $\Pi \cap \{i \mid (i,v) \in A\} = \emptyset$ **then**
9         $r^{\text{op}}_v \leftarrow \max\{c_i \mid i \in V \setminus \Pi$ such that $(i,v) \in A\}$
10   Set $r_{\min} = \min\{\max(r^{\text{op}}_v, r^{\text{mac}}_k) \mid v \in \Pi, k \in \mathcal{F}_v\}$ and let E be the set of pairs (v,k) with $v \in \Pi$ and $k \in \mathcal{F}_v$ such that $\max(r^{\text{op}}_v, r^{\text{mac}}_k) = r_{\min}$.
11     $(\hat{v},\hat{k}) \leftarrow \operatorname{argmin}\{r_{\min} + \psi_\alpha(p_{v,k}, g_k) \mid (v,k) \in E\}$.
12   Define $w_{\hat{v}} := \psi_\alpha(p_{\hat{v},\hat{k}}, g_{\hat{k}})$, $f_{\hat{v}} := \hat{k}$ and $c_{\hat{v}} := \max(r^{\text{op}}_{\hat{v}}, r^{\text{mac}}_{\hat{k}}) + w_{\hat{v}}$, and set $r^{\text{mac}}_{\hat{k}} \leftarrow c_{\hat{v}}$ and $g_{\hat{k}} \leftarrow g_{\hat{k}} + 1$.
13   **if** $|Q_{\hat{k}}| \neq 0$ **then**
14   Let $Q_{\hat{k}} = i_1, \dots, i_{|Q_{\hat{k}}|}$. Set $A \leftarrow A \cup \{(i_{|Q_{\hat{k}}|}, \hat{v})\}$.
15   Insert $\hat{v}$ at the end of $Q_{\hat{k}}$ and set $\Pi \leftarrow \Pi \setminus \{\hat{v}\}$.
16   CriticalPath($\mathcal{F}$, f, w, Q, G, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ).

The critical path in the directed graph $G=(V,A)$ can be computed with an adaptation [10, §22.2] of the Bellman-Ford algorithm; see Algorithm 2. In addition to the critical path $\mathcal{P}$, the algorithm returns a topological order $\mathcal{U}$ of the vertices of G and a vector τ of dimension $|\mathcal{F}|$. The vector τ stores, in element τ_k, the largest position in the list Q_k (list of operations assigned to machine k) that contains an operation in the critical path. These two elements are not used in the context of the present work, but they are useful information for developing neighborhoods in local search algorithms. Algorithm 2 has worst case time complexity $O(|V| + |A|)$. Since $|V| = O(|\mathcal{O}|)$ and $|A| = O(|\widehat{A}|+|\mathcal{O}|+|\mathcal{F}|)$, this complexity translates into $O(|\mathcal{O}|+|\widehat{A}|+|\mathcal{F}|)$. The topological order of Algorithm 2 is calculated with the help of Algorithm 3, which implements a depth-first search. Algorithm 3 computes, optionally, for each $v \in V$, the set $R_v^{\leftarrow}$ formed by the vertices w such that there exists in G a path from w to v, i.e. the vertices that can reach v. Algorithm 3 is recursive and its complexity is $O(|V| + |\widehat{A}|)$. The sets $R_v^{\leftarrow}$ are not used in the context of the present work, but, again, they contain valuable information for the development of local search strategies.

Algorithm 2: Computes a critical path $\mathcal{P}$ and its length ξ for a given graph $G = (V, A)$. In addition, if τ is present as an input parameter, determines the last critical operation in each machine.

Input: $\mathcal{F}$, f, w, Q, $G = (V, A)$, τ
Output: $\mathcal{U}$, $\mathcal{P}$, ξ, τ

function CriticalPath($\mathcal{F}$, f, w, Q, G, $\mathcal{U}$, $\mathcal{P}$, ξ, τ)
    Initialize $d_i \leftarrow -\infty$ for all $i \in V \setminus \{s\}$ and define $d_s := 0$ and $\pi_s := 0$.
    Initialize $\mathcal{V} \leftarrow \emptyset$ and $\mathcal{U}$ as an empty list and compute in $\mathcal{U}$ a topological sort of the vertices in V, by calling TopologicalSort+(G, $\mathcal{U}$, s, $\mathcal{V}$).
    for $\ell = 1, \dots, |V|$ **do**
        Let i be the ℓ-th operation in the topological order given by $\mathcal{U}$.
        for j such that $(i, j) \in A$ **do**
            if $d_j < d_i + w_i$ **then**
                $d_j \leftarrow d_i + w_i$ and $\pi_j \leftarrow i$.
    $\xi := d_t$
    Initialize $i \leftarrow \pi_t$, $\mathcal{P} \leftarrow \emptyset$, and $\tau_k \leftarrow 0$ for all $k \in \mathcal{F}$.
    do
        if $\tau_{f_i} = 0$ **then**
            Let Q_{f_i} be given by the sequence $i_1, \dots, i_{\ell-1}, i, i_{\ell+1}, \dots, i_{|Q_{f_i}|}$. Define $\tau_{f_i} := \ell$.
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{i\}$ and $i \leftarrow \pi_i$.
    while $i \neq s$

Algorithm 4 presents the constructive heuristic based on the ECT rule. The algorithm is very similar to Algorithm 1, except for one detail. In the constructive heuristic based on EST, we first compute the instant $r_{\min}$ which is the earliest instant at which an unscheduled operation could be initiated. All operation/machine pairs that could start at that instant are considered and the pair with the shortest processing time is selected. But since they would all start at instant $r_{\min}$, saying that the pair with the shortest processing time is chosen is the same as saying that the pair that ends earliest is selected. This is the idea that is taken to the extreme in the constructive heuristic based on the ECT rule: without limiting the choice to the operation/machine pairs that could start as early as possible, the operation/machine pair that

Algorithm 3: Computes a topological sort $\mathcal{U}$ of the vertices of $G = (V, A)$. In addition, if v and $\mathcal{R}_v^{\leftarrow}$ are present as an input parameter, computes the set $\mathcal{R}_v^{\leftarrow}$ of vertices that reaches v in $G = (V, A)$.

Input: $G = (V, A)$, $\mathcal{U}$, i, $\mathcal{V}$, v, $\mathcal{R}_v^{\leftarrow}$
Output: $\mathcal{U}$, $\mathcal{V}$, $\mathcal{R}_v^{\leftarrow}$

function TopologicalSort+(G, $\mathcal{U}$, i, $\mathcal{V}$, v, $\mathcal{R}_v^{\leftarrow}$)
  Set $\mathcal{V} \leftarrow \mathcal{V} \cup \{i\}$.
  for j such that $(i, j) \in A$ **do**
    if $j \notin \mathcal{V}$ **then**
      TopologicalSort+(G, $\mathcal{U}$, j, $\mathcal{V}$, v, $\mathcal{R}_v^{\leftarrow}$)
    if $i \notin \mathcal{R}_v^{\leftarrow}$ and $j \in \mathcal{R}_v^{\leftarrow}$ **then**
      set $\mathcal{R}_v^{\leftarrow} \leftarrow \mathcal{R}_v^{\leftarrow} \cup \{i\}$.
  Insert i at the beginning of $\mathcal{U}$.

will finish earliest is chosen, even if the processing of the operation does not start as early as possible. The worst case time complexity of Algorithm 4 is the same as that of Algorithm 1.

The EST-based heuristic gives priority to those pairs operations/machines that can start the earliest. At the beginning of the construction, this corresponds, roughly, to giving priority to all the first operations of each job, which are operations that have no precedents (operations 1, 3 and 7 in the example of Figure 1). Still, due to the intention of scheduling operations as early as possible, it is possible that preference is given to empty machines, building solutions that use several machines. By rapidly scheduling the first operations of each job, more operations come to have their precedents scheduled, increasing the number of possibilities (search space) in future iterations of the method. On the other hand, the heuristic based on the ECT rule chooses the operation/machine pairs that terminate the earliest, regardless of whether they are the ones that can start the earliest or not. Such a strategy can limit the number of operation/machine pairs available in future iterations, reducing the search space of the method. Moreover, the choice for the operation/machine pair that can finish earliest, combined with the learning effect, leads the method to schedule operations on machines that already have several operations assigned to them, since the higher the position in the machine, the shortest de processing (reduced by the positioned-based learning effect). This leads to the construction of solutions in which not all machines are used. Depending on the learning rate α considered and the density of the DAG of precedences of the instance at hand, one heuristic may be better than the other.

4 Benchmark instances

Tractability of the introduced models and performance of the proposed constructive heuristics will be evaluated with the 50 large-sized instances proposed in [4], that were introduced for the FJS scheduling problem with sequence flexibility but without learning effect. In addition to these large-sized instances, a new set with 60 smaller instances, using the generator described in [4], was generated. Instances whose name starts with "Y" correspond to instances in which

Algorithm 4: Computes the solution graph $G = (V, A)$, f, Q, and w by using ECT dispatching rule. Then, in G, it computes the largest path $\mathcal{P}$ from s to t and its length $C_{\max}$.

Input: $\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$

Output: f, w, Q, $G = (V, A)$, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ

function ECT($\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$, f, w, Q, G, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ)
- Set $A \leftarrow \widehat{A} \cup \{(s, j) \mid (\cdot, j) \notin \widehat{A}\} \cup \{(i, t) \mid (i, \cdot) \notin \widehat{A}\}$ and define $V := \mathcal{O} \cup \{s, t\}$ and $G = (V, A)$.
- Set $r_v^{\text{op}} \leftarrow +\infty$ and define $r_s^{\text{op}} := 0$, $w_s := w_t := 0$, and $c_s := 0$.
- Set $r_k^{\text{mac}} \leftarrow 0$ and $g_k \leftarrow 1$ for all $k \in \mathcal{F}$.
- Initialize $\Pi \leftarrow V \setminus \{s, t\}$ as the set of non-scheduled operations, and Q_k as an empty list for all $k \in \mathcal{F}$.
- **while** $\Pi \neq \emptyset$ **do**
  - **for** $v \in \Pi$ **do**
    - **if** $\Pi \cap \{i \mid (i, v) \in A\} = \emptyset$ **then**
      - $r_v^{\text{op}} \leftarrow \max\{c_i \mid i \in V \setminus \Pi \text{ such that } (i, v) \in A\}$
  - $(\hat{v}, \hat{k}) \leftarrow \operatorname{argmin}\{\max(r_v^{\text{op}}, r_k^{\text{mac}}) + \psi_\alpha(p_{v,k}, g_k) \mid v \in \Pi, k \in \mathcal{F}_v\}$.
  - Define $w_{\hat{v}} := \psi_\alpha(p_{\hat{v},\hat{k}}, g_{\hat{k}})$, $f_{\hat{v}} := \hat{k}$ and $c_{\hat{v}} := \max(r_{\hat{v}}^{\text{op}}, r_{\hat{k}}^{\text{mac}}) + w_{\hat{v}}$ and set $r_{\hat{k}}^{\text{mac}} \leftarrow c_{\hat{v}}$ and $g_{\hat{k}} \leftarrow g_{\hat{k}} + 1$.
  - **if** $|Q_{\hat{k}}| \neq 0$ **then**
    - Let $Q_{\hat{k}} = i_1, \dots, i_{|Q_{\hat{k}}|}$. Set $A \leftarrow A \cup \{(i_{|Q_{\hat{k}}|}, \hat{v})\}$.
  - Insert $\hat{v}$ at the end of $Q_{\hat{k}}$ and set $\Pi \leftarrow \Pi \setminus \{\hat{v}\}$.
- CriticalPath($\mathcal{F}$, f, w, Q, G, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ).

DAGs that represent the operations' precedences are Y-shaped (like the DAG in the top of Figure 1); while instances whose name starts with "DA" correspond to instances in which DAGs that represent the operations precedences are arbitrary DAGs (like the DAG in the bottom of Figure 1). The former will be called instances of Y-type and the latter will be called instances of DA-type from now on.

For a given instance, we define measures ω_1 and ω_2 of the routes' nonlinearity and the routing flexibility, respectively. Both measures are between 0 and 1 and, the larger their values, the larger the flexibility they represent. Moreover, the larger the flexibility, the larger the search space and, in consequence, the harder the instance. Let $D^+ = (\mathcal{O}, \widehat{A}^+)$ be the transitive closure of the precedences DAG $D = (\mathcal{O}, \widehat{A})$. Let n_κ be the number of operations of job κ and let a_κ the number of arcs in the D^+ among them. Then, $a_\kappa^{\min} \leq a_\kappa \leq a_\kappa^{\max}$, where $a^{\min}(n_\kappa) = n_\kappa - 1$ and $a^{\max}(n_\kappa) = n_\kappa(n_\kappa - 1)/2$. Therefore

$$\omega_1^\kappa := 1 - \frac{a_\kappa - a^{\min}(n_\kappa)}{a^{\max}(n_\kappa) - a^{\min}(n_\kappa)},$$

is such that $\omega_1^\kappa \in [0, 1]$ represents the routes' nonlinearity of the operations of job κ. The arithmetic mean among all n jobs of the instance, given by

$$\omega_1 = \frac{1}{n} \sum_{\kappa=1}^{n} \omega_1^{\kappa},$$

is also between 0 and 1 and represent the degree of the instance routes' nonlinearity. The larger ω_1, the larger the search space and, therefore, the more difficult the instance. (Of course, any other type of average could be used in the definition of ω_1.)

In a similar way, we define the routing flexibility measure ω_2. It is easy to see that $|\mathcal{O}| \leq \sum_{i \in \mathcal{O}} |\mathcal{F}_i| = \sum_{k \in \mathcal{F}} |\mathcal{O}_k| \leq |\mathcal{O}||\mathcal{F}|$. Therefore,

$$\omega_2 = \frac{\sum_{i \in \mathcal{O}} |\mathcal{F}_i| - |\mathcal{O}|}{|\mathcal{O}||\mathcal{F}| - |\mathcal{O}|},$$

is between 0 and 1. The closer to 1, the larger the search space and, therefore, the harder the instance.

Tables 1 and 2 provide a comprehensive overview of the instances main characteristics. In the tables, $|\mathcal{O}|$ is the number of operations, $|\mathcal{F}|$ is the number of machines, n is the number of jobs (connected components in the precedence constraints DAG $D = (\mathcal{O}, \widehat{A})$), $|\widehat{A}|$ is the number of precedence constraints, $\sum_{i \in \mathcal{O}} |\mathcal{F}_i|$ is the number of "true" entries in the logical matrix of dimension $|\mathcal{O}| \times |\mathcal{F}|$ whose (i, k) says whether operation i can be processed by machine k, and ω_1 and ω_2 are the measures of the routes' nonlinearity and routing flexibility described in the previous paragraph. In addition, for the MILP and the CP models, the tables show the number of variables and constraints.

In the small-sized instances of Table 1, the number of binary variables of the MILP models and the number of interval variables of the CP models go up to almost 1,000; while in both models the number of constraints goes up to 13,000. On the other hand, in the large-sized instances of Table 2, the number of binary variables of the MILP models and the number of interval variables of the CP models go up to almost 73,000; while in both models the number of constraints goes up to 4,000,000. Moreover, for each instance, the number of binary variables in its MILP model is very similar to the number of interval variables in its CP model and the two models also have a very similar number of constraints.

5 Numerical experiments

In this section we present numerical experiments. First, we wish to evaluate the two introduced constructive heuristics. Second, we wish to assess the correctness of the MILP and CP models and attempt to infer which of the two, or rather which of the exact commercial solvers applied to each of them, is more effective in finding proven optimal solutions. Third, we wish to determine the usefulness of providing a feasible solution to the exact solvers. It should be noted that all efforts are to build a set of test instances with proven optimal solutions. The models and constructive heuristics presented in this paper are intended to contribute in that respect and are not intended to construct a solution method per se, for a known difficult problem. In all cases we considered the 110 instances introduced in Section 4 with the learning rate $\alpha \in \{0.1, 0.2, 0.3\}$ for a total of 330 instances.

Main instance characteristics								MILP formulation			CP Optimizer formulation	
Instance	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{O}\rvert$	n	$\lvert\widehat{A}\rvert$	$\sum_{i\in\mathcal{O}}\mathcal{F}_i$	ω_1	ω_2	#binary variables	#continous variables	#constraints	#interval variables	#constraints
miniDAFJS01	5	14	2	14	44	0.54	0.54	392	78	3,754	406	3,564
miniDAFJS02	5	11	2	9	35	0.57	0.55	251	63	2,018	262	1,867
miniDAFJS03	5	10	2	12	31	0.50	0.53	197	57	1,435	207	1,301
miniDAFJS04	5	9	2	9	29	0.25	0.56	175	53	1,234	184	1,109
miniDAFJS05	5	15	2	15	38	0.39	0.38	310	74	2,881	325	2,714
miniDAFJS06	5	14	2	14	46	0.20	0.57	448	80	4,742	462	4,544
miniDAFJS07	5	11	2	10	34	0.23	0.52	240	62	1,924	251	1,777
miniDAFJS08	5	9	2	8	24	0.42	0.42	122	48	770	131	665
miniDAFJS09	5	15	2	13	47	0.63	0.53	465	83	5,018	480	4,815
miniDAFJS10	5	11	2	10	32	0.28	0.48	208	60	1,530	219	1,391
miniDAFJS11	5	18	2	16	43	0.45	0.35	409	85	4,371	427	4,181
miniDAFJS12	5	12	2	12	38	0.55	0.54	298	68	2,602	310	2,438
miniDAFJS13	5	10	2	8	27	0.58	0.43	149	53	979	159	861
miniDAFJS14	5	14	2	12	44	0.50	0.54	394	78	3,800	408	3,610
miniDAFJS15	5	11	2	11	35	0.40	0.55	261	63	2,242	272	2,091
miniDAFJS16	5	13	2	12	42	0.18	0.56	358	74	3,302	371	3,121
miniDAFJS17	5	11	2	11	30	0.13	0.43	194	58	1,491	205	1,360
miniDAFJS18	5	11	2	10	34	0.38	0.52	234	62	1,798	245	1,651
miniDAFJS19	5	12	2	10	36	0.30	0.50	272	66	2,338	284	2,182
miniDAFJS20	5	13	2	11	40	0.54	0.52	350	72	3,447	363	3,274
miniDAFJS21	5	19	3	20	60	0.52	0.54	730	104	9,298	749	9,039
miniDAFJS22	5	18	3	18	57	0.22	0.54	665	99	8,223	683	7,977
miniDAFJS23	5	21	3	18	66	0.53	0.54	874	114	11,934	895	11,649
miniDAFJS24	5	18	3	17	56	0.36	0.53	642	98	7,809	660	7,567
miniDAFJS25	5	17	3	17	51	0.12	0.50	535	91	6,024	552	5,803
miniDAFJS26	5	17	3	15	53	0.36	0.53	573	93	6,560	590	6,331
miniDAFJS27	5	19	3	16	57	0.46	0.50	721	101	10,155	740	9,908
miniDAFJS28	5	16	3	15	48	0.54	0.50	468	86	4,871	484	4,663
miniDAFJS29	5	14	3	15	47	0.33	0.59	451	81	4,634	465	4,432
miniDAFJS30	5	22	3	21	61	0.32	0.44	791	111	11,020	813	10,754
miniYFJS01	7	16	4	12	54	0.25	0.40	420	94	3,554	436	3,322
miniYFJS02	7	16	4	12	44	0.33	0.29	294	84	2,298	310	2,106
miniYFJS03	7	16	4	12	45	0.08	0.30	333	85	2,915	349	2,719
miniYFJS04	7	16	4	12	53	0.25	0.39	439	93	4,083	455	3,855
miniYFJS05	7	16	4	12	55	0.50	0.41	457	95	4,249	473	4,013
miniYFJS06	7	16	4	12	51	0.42	0.36	407	91	3,773	423	3,553
miniYFJS07	7	16	4	12	48	0.17	0.33	350	88	2,912	366	2,704
miniYFJS08	7	16	4	12	49	0.50	0.34	363	89	3,037	379	2,825
miniYFJS09	7	16	4	12	51	0.33	0.36	399	91	3,569	415	3,349
miniYFJS10	7	16	4	12	59	0.17	0.45	509	99	4,767	525	4,515
miniYFJS11	7	20	5	15	56	0.13	0.30	464	104	4,241	484	3,997
miniYFJS12	7	20	5	15	68	0.13	0.40	716	116	8,363	736	8,071
miniYFJS13	7	20	5	15	69	0.40	0.41	723	117	8,272	743	7,976
miniYFJS14	7	20	5	15	59	0.60	0.33	509	107	4,772	529	4,516
miniYFJS15	7	20	5	15	53	0.33	0.28	429	101	3,944	449	3,712
miniYFJS16	7	20	5	15	63	0.40	0.36	617	111	6,838	637	6,566
miniYFJS17	7	20	5	15	57	0.27	0.31	485	105	4,576	505	4,328
miniYFJS18	7	20	5	15	51	0.07	0.26	395	99	3,514	415	3,290
miniYFJS19	7	20	5	15	58	0.47	0.32	512	106	5,031	532	4,779
miniYFJS20	7	20	5	15	62	0.53	0.35	638	110	7,295	658	7,027
miniYFJS21	7	24	6	18	72	0.28	0.33	756	128	8,466	780	8,154
miniYFJS22	7	24	6	18	82	0.28	0.40	996	138	12,962	1,020	12,610
miniYFJS23	7	24	6	18	78	0.17	0.38	908	134	11,280	932	10,944
miniYFJS24	7	24	6	18	62	0.28	0.26	588	118	6,136	612	5,864
miniYFJS25	7	24	6	18	76	0.17	0.36	836	132	9,686	860	9,358
miniYFJS26	7	24	6	18	67	0.33	0.30	693	123	7,841	717	7,549
miniYFJS27	7	24	6	18	81	0.39	0.40	951	137	11,709	975	11,361
miniYFJS28	7	24	6	18	67	0.22	0.30	661	123	7,001	685	6,709
miniYFJS29	7	24	6	18	80	0.39	0.39	934	136	11,536	958	11,192
miniYFJS30	7	24	6	18	72	0.44	0.33	800	128	9,756	824	9,444

Table 1: Main features of the proposed sixty small-sized instances.

The experiments were carried out in an Intel i9-12900K (12th Gen) processor operating at 5.200GHz and 128 GB of RAM. The constructive heuristics were implemented in C++ programming language. Models were solved using IBM ILOG CPLEX Optimization Studio version 22.1, using default parameters, with concert library and C++. The code was compiled using g++

	Main instance characteristics							MILP formulation			CP Optimizer formulation	
Instance	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{O}\rvert$	n	$\lvert\widehat{A}\rvert$	$\sum_{i\in\mathcal{O}}\mathcal{F}_i$	ω_1	ω_2	#binary variables	#continous variables	#constraints	#interval variables	#constraints
DAFJS01	5	26	4	26	82	0.54	0.54	1,358	140	23,102	1,384	22,748
DAFJS02	5	25	4	23	79	0.45	0.54	1,273	135	21,348	1,298	21,007
DAFJS03	10	55	4	52	279	0.32	0.45	7,849	400	223,911	7,904	222,740
DAFJS04	10	43	4	40	220	0.25	0.46	4,960	317	115,208	5,003	114,285
DAFJS05	5	39	6	34	104	0.35	0.42	2,242	188	50,228	2,281	49,773
DAFJS06	5	44	6	41	136	0.38	0.52	3,724	230	103,253	3,768	102,665
DAFJS07	10	85	6	82	431	0.30	0.45	18,695	612	817,681	18,780	815,872
DAFJS08	10	85	6	82	403	0.31	0.42	16,357	584	670,739	16,442	669,042
DAFJS09	5	45	8	42	135	0.40	0.50	3,755	231	107,667	3,800	107,082
DAFJS10	5	58	8	52	168	0.40	0.47	5,764	290	201,720	5,822	200,990
DAFJS11	10	113	8	108	534	0.40	0.41	28,648	771	1,546,720	28,761	1,544,471
DAFJS12	10	117	8	114	603	0.49	0.46	36,513	848	2,223,141	36,630	2,220,612
DAFJS13	5	62	10	55	193	0.41	0.53	7,511	323	295,582	7,573	294,748
DAFJS14	5	69	10	62	206	0.37	0.50	8,578	350	361,926	8,647	361,033
DAFJS15	10	120	10	117	595	0.32	0.44	35,811	846	2,184,302	35,931	2,181,802
DAFJS16	10	120	10	114	602	0.33	0.45	36,344	853	2,203,066	36,464	2,200,538
DAFJS17	5	82	12	77	246	0.43	0.50	12,244	416	617,041	12,326	615,975
DAFJS18	5	74	12	64	231	0.41	0.53	10,785	385	509,435	10,859	508,437
DAFJS19	7	70	8	66	283	0.34	0.51	11,507	431	471,949	11,577	470,747
DAFJS20	7	92	10	87	361	0.36	0.49	18,709	553	976,026	18,801	974,490
DAFJS21	7	107	12	102	425	0.38	0.50	25,853	647	1,577,783	25,960	1,575,976
DAFJS22	7	116	12	109	450	0.39	0.48	29,296	690	1,932,281	29,412	1,930,365
DAFJS23	9	76	8	71	367	0.31	0.48	15,103	529	628,938	15,179	627,394
DAFJS24	9	92	8	87	463	0.31	0.50	23,893	657	1,238,922	23,985	1,236,978
DAFJS25	9	123	10	119	619	0.31	0.50	42,753	875	2,967,430	42,876	2,964,831
DAFJS26	9	119	10	116	606	0.34	0.51	41,026	854	2,794,800	41,145	2,792,257
DAFJS27	9	127	12	118	625	0.27	0.49	43,461	889	3,029,045	43,588	3,026,418
DAFJS28	10	91	8	89	457	0.32	0.45	21,065	650	980,790	21,156	978,871
DAFJS29	10	95	8	94	468	0.34	0.44	22,450	669	1,109,378	22,545	1,107,411
DAFJS30	10	98	10	94	509	0.20	0.47	26,059	716	1,344,045	26,157	1,341,911
YFJS01	7	40	4	36	104	0.10	0.27	1,824	192	38,286	1,864	37,830
YFJS02	7	40	4	36	104	0.17	0.27	1,568	192	24,498	1,608	24,042
YFJS03	7	24	6	18	63	0.28	0.27	611	119	6,561	635	6,285
YFJS04	7	28	7	21	71	0.19	0.26	813	135	10,260	841	9,948
YFJS05	7	32	8	24	81	0.33	0.26	1,003	153	13,417	1,035	13,061
YFJS06	7	36	9	27	95	0.19	0.27	1,365	175	21,214	1,401	20,798
YFJS07	7	36	9	27	93	0.26	0.26	1,279	173	18,588	1,315	18,180
YFJS08	12	36	9	27	100	0.26	0.16	888	185	8,879	924	8,443
YFJS09	12	36	9	27	219	0.22	0.46	4,079	304	78,474	4,115	77,562
YFJS10	12	40	10	30	113	0.17	0.17	1,169	206	13,593	1,209	13,101
YFJS11	10	50	10	40	134	0.22	0.19	1,860	245	27,378	1,910	26,792
YFJS12	10	50	10	40	133	0.12	0.18	1,915	244	30,085	1,965	29,503
YFJS13	10	50	10	40	137	0.15	0.19	1,895	248	27,177	1,945	26,579
YFJS14	26	221	13	208	641	0.24	0.08	16,603	1,110	454,179	16,824	451,394
YFJS15	26	221	13	208	648	0.23	0.08	16,620	1,117	441,752	16,841	438,939
YFJS16	26	221	13	208	633	0.13	0.07	16,037	1,102	424,253	16,258	421,500
YFJS17	26	289	17	272	1328	0.15	0.14	68,502	1,933	3,572,342	68,791	3,566,741
YFJS18	26	289	17	272	1362	0.15	0.15	72,354	1,967	3,901,378	72,643	3,895,641
YFJS19	26	289	17	272	1347	0.20	0.15	70,527	1,952	3,737,737	70,816	3,732,060
YFJS20	26	289	17	272	1343	0.12	0.15	70,371	1,948	3,742,757	70,660	3,737,096

Table 2: Main features of the fifty large-sized instances from [4].

10.2.1. Benchmark instances and code are available at `https://github.com/kennedy94/FJS`.

5.1 Experiments with the constructive heuristics

In this subsection, we evaluate the two constructive heuristics. Tables 3 and 4 show the results. For each instance, the lowest makespan, among the solutions found by the two constructive heuristics, is shown in bold. In all instances, the constructive heuristics take less than 0.001 seconds of CPU time to build a solution. Considering the small-sized instances in Table 3, depending on the instance, there may be a significant difference between the solutions found by one and the other constructive heuristic. However, on average, the two heuristics behave basically indistinguishably. For the large-sized instances in Table 4 the comparison is a bit different: there is a clear advantage of the EST constructive heuristic in the DA-type instances, while on the other hand there is a clear advantage of the ECT constructive heuristic in the Y-type instances. It is worth noting that in the small-sized instances there is no clear differentiation between the routes' nonlinearity sparsity measure ω_1 of the DA-type and the Y-type instances (see Table 1). On the other hand, in the large-sized instances, Y-type instances have a value of ω_1 clearly lower than the value of ω_1 of DA-type instances. Summarizing, as mentioned at the end of Section 3, the greedy strategy of ECT of choosing the operation/machine pair that terminates first seems to compensate in situations where, because there is already little routes' nonlinearity, the greedy choice does not cause a large decrease of the search space.

5.2 Solving the proposed models with a commercial solver

In this section, we apply an exact solver to the 330 instances considered and evaluate the quality of the solutions found within a given CPU time limit, depending on whether we provide the solution found by the constructive heuristics as an initial solution or not. Since there is no clear winner between the two constructive heuristics and their execution time is negligible, we run both and give the best of the two as initial solution to the exact solver. In the tables and figures, the solvers' runs that receive as input a solution computed by one of the constructive heuristics are identified with the expression "warm start".

A solution is reported as optimal by the solvers when

$$\text{absolute gap} = \text{best feasible solution} - \text{best lower bound} \leq \epsilon_{\text{abs}} \tag{23}$$

or

$$\text{relative gap} = \frac{|\text{best feasible solution} - \text{best lower bound}|}{10^{-10} + |\text{best feasible solution}|} \leq \epsilon_{\text{rel}}, \tag{24}$$

where, by default, $\epsilon_{\text{abs}} = 10^{-6}$ and $\epsilon_{\text{rel}} = 10^{-4}$, and "best feasible solution" means the smallest value of the makespan related to a feasible solution generated by the method. Since the optimal makespan of the instances considered in this work always assumes an integer value, we choose to use $\epsilon_{\text{abs}} = 1 - 10^{-6}$ and $\epsilon_{\text{rel}} = 0$. The choice $\epsilon_{\text{rel}} = 0$ avoids premature stops in a solution that may not be optimal. The choice $\epsilon_{\text{abs}} = 1 - 10^{-6}$ allows to stop early when a relative gap smaller than 1 clearly indicates that the optimal solution has already been found. A CPU time limit of 1 hour was imposed. All other solvers parameters were used with their default values.

Tables 5, 6, 7, 8, 9, and 10 show the results in detail when the warm start is not taken into account. Tables 5, 6, and 7 correspond to the small-sized instances with α equal to 0.1, 0.2, and 0.3, respectively; while Tables 8, 9, and 10 show the same thing for the large-sized instances. Tables 11–16 show the results when the warm start is taken into account. The tables show the information related to the resolution of the MILP and CP models. When a single number appears in the "makespan" column, it means that a provably optimal solution with that makespan value was found. When instead of a number an expression of the form $[A, B]\,C\%$ appears, it means that a feasible solution was found with value B for the makespan, value A for a lower bound on the makespan, and gap C equal to $100(B - A)/B$. As measures of computational effort, for the MILP solver the tables show the number of iterations, the number of nodes explored in

instance	$\alpha = 0.1$		$\alpha = 0.2$		$\alpha = 0.3$	
	EST	ECT	EST	ECT	EST	ECT
miniDAFJS01	**23,264**	23,948	**22,156**	23,199	**20,757**	21,535
miniDAFJS02	**23,242**	23,860	**22,161**	22,775	**21,152**	21,461
miniDAFJS03	**18,363**	**18,363**	**17,972**	**17,972**	**17,621**	**17,621**
miniDAFJS04	**21,690**	**21,690**	21,233	**21,121**	20,824	**20,391**
miniDAFJS05	24,279	**22,598**	21,815	**20,826**	20,623	**19,253**
miniDAFJS06	23,726	**23,370**	23,781	**22,193**	**21,148**	21,480
miniDAFJS07	**28,644**	33,413	27,551	**25,088**	26,600	**24,767**
miniDAFJS08	**19,878**	25,100	**18,857**	22,921	**17,927**	20,943
miniDAFJS09	**25,425**	27,806	**23,830**	25,400	**22,398**	23,295
miniDAFJS10	22,847	**21,563**	21,730	**21,021**	20,349	**20,176**
miniDAFJS11	35,166	**33,911**	32,214	**31,269**	29,739	**28,718**
miniDAFJS12	**20,342**	21,435	**19,624**	20,370	**19,094**	19,394
miniDAFJS13	**18,313**	18,681	**17,143**	17,181	16,077	**15,425**
miniDAFJS14	**26,503**	27,525	**24,958**	25,919	**23,552**	24,467
miniDAFJS15	24,961	**22,472**	23,710	**21,969**	22,545	**20,524**
miniDAFJS16	25,845	**25,691**	24,699	**24,593**	26,750	**23,797**
miniDAFJS17	**21,070**	21,144	**20,519**	**20,519**	**19,941**	**19,941**
miniDAFJS18	19,716	**19,308**	**18,829**	19,133	**17,784**	18,395
miniDAFJS19	**21,293**	21,329	**20,107**	20,340	**19,030**	19,426
miniDAFJS20	**23,570**	26,421	21,759	**21,286**	20,140	**19,587**
miniDAFJS21	25,764	**25,020**	24,040	**22,689**	23,171	**21,299**
miniDAFJS22	**28,122**	31,892	**26,289**	29,188	**24,671**	26,829
miniDAFJS23	**27,566**	33,389	**25,592**	30,242	**23,847**	28,000
miniDAFJS24	**26,932**	27,581	29,407	**24,500**	27,040	**22,524**
miniDAFJS25	**23,370**	25,879	**22,613**	24,209	**21,923**	22,976
miniDAFJS26	26,408	**25,974**	24,476	**23,630**	22,771	**21,613**
miniDAFJS27	**28,084**	31,060	**25,429**	26,652	**23,089**	23,595
miniDAFJS28	**28,836**	30,947	**26,587**	27,477	26,831	**24,429**
miniDAFJS29	27,569	**21,151**	25,923	**19,789**	24,444	**18,560**
miniDAFJS30	27,870	**27,249**	25,318	**24,491**	**21,421**	22,071
wins	19	13	16	16	16	16
mean	24,621.93	25,325.67	23,344.07	23,265.40	22,108.63	21,749.73

instance	$\alpha = 0.1$		$\alpha = 0.2$		$\alpha = 0.3$	
	EST	ECT	EST	ECT	EST	ECT
miniYFJS01	40,456	**35,243**	38,132	**34,443**	36,008	**33,697**
miniYFJS02	34,009	**28,688**	31,940	**27,557**	27,987	**25,969**
miniYFJS03	**62,704**	69,005	**57,950**	60,302	53,679	**52,865**
miniYFJS04	32,133	**25,394**	30,048	**24,669**	24,428	**24,017**
miniYFJS05	29,009	**27,650**	26,828	**26,135**	27,047	**24,743**
miniYFJS06	49,001	**30,952**	43,387	**29,080**	40,871	**27,366**
miniYFJS07	**51,782**	57,652	**47,743**	53,702	**44,128**	50,110
miniYFJS08	**35,336**	51,013	**33,178**	40,884	**31,133**	40,567
miniYFJS09	**38,626**	39,463	40,411	**37,593**	38,480	**34,357**
miniYFJS10	**32,975**	34,363	**30,651**	31,742	**28,513**	29,397
miniYFJS11	51,600	**51,212**	48,172	**47,079**	45,079	**43,254**
miniYFJS12	**38,889**	39,024	**36,036**	36,691	**33,495**	35,223
miniYFJS13	38,448	**35,177**	34,046	**31,720**	**26,536**	28,751
miniYFJS14	**41,560**	42,764	**39,254**	39,862	**37,157**	37,177
miniYFJS15	**57,017**	58,693	**51,504**	**51,504**	**46,583**	**46,583**
miniYFJS16	39,097	**35,831**	36,810	**33,285**	34,799	**30,576**
miniYFJS17	**53,052**	62,576	**48,745**	57,600	**44,916**	50,137
miniYFJS18	**37,217**	39,284	34,053	**31,911**	31,154	**29,764**
miniYFJS19	46,640	**46,478**	41,729	**41,451**	42,825	**37,034**
miniYFJS20	**42,408**	42,465	39,852	**39,600**	37,509	**37,063**
miniYFJS21	50,076	**44,016**	45,541	**40,765**	40,884	**35,905**
miniYFJS22	**43,722**	47,389	40,428	**32,485**	31,887	**30,212**
miniYFJS23	**53,046**	59,584	**49,209**	52,628	**45,823**	46,589
miniYFJS24	**42,268**	45,974	**39,883**	41,759	**37,651**	37,998
miniYFJS25	**46,936**	68,698	**40,933**	61,260	**35,966**	54,247
miniYFJS26	55,031	**55,022**	49,428	**49,412**	**53,604**	54,006
miniYFJS27	43,319	**41,060**	39,256	**36,561**	35,655	**33,799**
miniYFJS28	46,385	**43,232**	42,230	**40,325**	39,363	**37,547**
miniYFJS29	**48,806**	55,473	43,757	**41,834**	39,279	**39,146**
miniYFJS30	**49,458**	52,043	**45,051**	46,003	41,116	**40,752**
wins	17	13	12	19	12	19
mean	44,366.87	45,513.93	40,872.83	40,661.40	37,785.17	37,628.37

Table 3: Makespan values for the small-sized set of instance solved by constructive heuristics.

instance	$\alpha = 0.1$		$\alpha = 0.2$		$\alpha = 0.3$	
	EST	ECT	EST	ECT	EST	ECT
DAFJS01	**29,769**	41,358	**28,920**	31,657	**22,616**	28,222
DAFJS02	**32,467**	33,155	29,089	**28,154**	**26,183**	26,949
DAFJS03	**53,688**	53,834	**48,555**	51,717	**43,964**	45,501
DAFJS04	**54,082**	54,150	**48,461**	48,503	**43,281**	44,094
DAFJS05	52,651	**45,862**	**44,790**	44,919	**38,237**	38,457
DAFJS06	**51,925**	56,012	**44,228**	46,403	**38,007**	41,229
DAFJS07	**57,193**	59,491	51,019	**50,802**	**41,285**	44,518
DAFJS08	**62,159**	67,107	53,998	**53,439**	**46,323**	47,173
DAFJS09	**48,680**	60,565	48,833	**42,537**	**41,418**	42,518
DAFJS10	**58,695**	60,374	**46,257**	49,140	**38,771**	41,125
DAFJS11	**67,594**	92,639	**56,847**	75,992	**47,805**	56,547
DAFJS12	70,287	**68,322**	58,100	**55,575**	**45,735**	49,991
DAFJS13	63,386	**61,794**	**51,317**	53,486	**42,319**	44,724
DAFJS14	83,362	**76,862**	**62,724**	63,887	**49,459**	50,737
DAFJS15	78,413	**72,288**	**54,353**	69,279	**49,206**	54,179
DAFJS16	78,289	**76,204**	**65,550**	72,885	57,213	**56,727**
DAFJS17	**73,219**	84,719	**63,177**	65,307	**54,527**	54,887
DAFJS18	82,129	**78,862**	**61,831**	67,753	**51,823**	52,325
DAFJS19	**66,412**	67,169	**58,001**	63,933	49,746	**46,880**
DAFJS20	**78,778**	80,562	**63,588**	68,034	**51,686**	54,531
DAFJS21	**78,320**	83,933	**63,202**	66,878	**53,010**	54,906
DAFJS22	77,853	**70,892**	**56,115**	62,199	**45,005**	49,723
DAFJS23	**49,969**	53,123	**47,616**	50,639	**39,249**	41,475
DAFJS24	**57,411**	62,038	**49,019**	50,345	**44,851**	46,119
DAFJS25	89,248	**82,055**	**66,262**	67,533	**51,964**	60,266
DAFJS26	**81,480**	83,635	75,230	**72,226**	**57,182**	60,803
DAFJS27	**81,470**	88,629	**64,797**	71,903	**58,138**	58,145
DAFJS28	**62,568**	64,560	**52,639**	53,110	**42,898**	46,294
DAFJS29	74,841	**72,938**	**59,792**	66,900	**51,606**	58,809
DAFJS30	**61,147**	70,062	**55,015**	67,245	**43,859**	47,605
wins	20	10	24	6	28	2
mean	65,249.50	67,439.80	54,310.83	57,746.00	45,578.87	48,181.97

instance	$\alpha = 0.1$		$\alpha = 0.2$		$\alpha = 0.3$	
	EST	ECT	EST	ECT	EST	ECT
YFJS01	**87,203**	106,117	**84,152**	92,107	**70,402**	80,419
YFJS02	87,462	**81,579**	73,957	**66,853**	68,075	**61,111**
YFJS03	42,457	**40,197**	**35,380**	37,159	**31,680**	33,077
YFJS04	**47,467**	50,724	**43,614**	44,467	39,249	**38,898**
YFJS05	**46,138**	55,871	**40,851**	49,893	**41,586**	45,012
YFJS06	53,210	**52,487**	**48,564**	54,660	**46,811**	47,173
YFJS07	63,320	**54,457**	56,313	**51,261**	**39,896**	44,004
YFJS08	51,818	**49,626**	47,218	**44,074**	**39,325**	39,469
YFJS09	38,836	**28,354**	36,123	**26,072**	32,120	**24,027**
YFJS10	**42,583**	59,808	**40,658**	52,811	**39,607**	48,821
YFJS11	65,011	**59,356**	58,106	**51,469**	51,943	**45,051**
YFJS12	**70,830**	74,978	**59,898**	62,299	53,275	**52,398**
YFJS13	53,601	**50,805**	48,729	**45,084**	41,303	**40,119**
YFJS14	151,365	**129,428**	116,157	**109,469**	111,313	**92,457**
YFJS15	152,375	**138,196**	120,600	**112,424**	107,006	**96,547**
YFJS16	144,976	**127,055**	131,230	**106,855**	118,515	**92,811**
YFJS17	133,982	**109,112**	110,203	**85,736**	98,045	**73,682**
YFJS18	154,214	**133,703**	121,563	**99,429**	104,338	**87,059**
YFJS19	133,142	**107,055**	110,125	**89,561**	91,287	**74,431**
YFJS20	137,326	**97,868**	104,036	**91,958**	91,229	**72,481**
wins	5	15	7	13	7	13
mean	87,865.80	80,338.80	74,373.85	68,682.05	65,850.25	59,452.35

Table 4: Makespan values for the testbed set of instance solved by constructive heuristics.

the branch-and-bound search tree, and the CPU time in seconds. For the CP solver, the tables show the number of branches and the CPU time in seconds. If no information is displayed for a particular instance, it means that the solver was unable to find a feasible solution within the specified CPU time limit. In Tables 11–16, which show results with warm start, the symbol † next to the optimal or best value found means that the exact method returned exactly the same solution computed with the constructive heuristic and given as initial solution. The information from Tables 5–16 is presented graphically in Figure 6.

Let's look at the small-sized instances first. Without warm start, the exact methods found provably optimal solutions for 168 instances of the MILP model and 169 instances of the CP model (out of a total of 180 small-sized instances). In the remaining cases, the gaps for the MILP model instances were between 0.17% and 23.32%, while for the CP model instances the gaps were between 29.44% and 47.55%. It is worth noting that in the few cases without a proven optimal solution, there is a slight advantage in the best solution found for the CP model instances and a slight advantage in the lower bounds found for the MILP model instances. In the instances where a proven solution was found by solving both the MILP model and the CP model, the CP solver was on average 12.09% faster than the MILP solver. When the constructive heuristics solution is made available to the exact solvers, the number of proven optimal solutions found hardly changes (still the same number for MILP model instances and one less for CP model instances, not necessarily the same as those solved without the warm start). However, for MILP model instances where a proven optimal solution is found both with and without warm start, the use of warm start reduces the solution time by 32.52% in average. This reduction is 4.44% for the CP model solver. One way or another, whether using the MILP model or the CP model, with or without warm start, it was possible to find provably optimal solutions in 176 (out of 180) small-sized instances.

The analysis of the 150 large-sized instances is a little different. Without a warm start, the exact methods were able to find proven optimal solutions for only 7 instances of the MILP model and 5 instances of the CP model. In 70 MILP model instances it was possible to find feasible solutions with gaps between 13.94% and 90.47%, while feasible solutions with gaps between 9.30% and 86.10% were found in 145 instances of the CP model. In 73 instances of the MILP model, not a single feasible solution was found. In the 70 instances in which feasible solutions from both the MILP and CP models were found, the solutions found using the CP model were on average 68.85% better. It was after observing these results that the idea arose to develop constructive heuristics to test the warm start and consider a set of smaller instances.

When the solution of the constructive heuristics is fed to the exact solver, 6 provably optimal solutions and 144 feasible solutions are obtained with gaps between 5.65% and 64.36% for MILP model instances. For the CP model instances, the same number of provably optimal and feasible solutions are found, with gaps between 15.43% and 81.06% for the feasible ones. In those instances where a proven optimal solution is found without and with warm start, warm start *increases* the computational cost of solving the MILP and CP models by 9.69% and 25.99%, respectively. On the other hand, in the MILP model instances in which a feasible solution was found without the use of warm start, the use of warm start improved the quality of the feasible solution found by 49.46%. For the CP model instances, this improvement was 11.53%. In the 144 instances in which feasible solutions from both the MILP and CP models were found, the solutions found using the CP model were on average 6.82% better. The question remains as

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
miniDAFJS01	22,875	278,358	2,630	20.9	22,875	134,773	5.5
miniDAFJS02	22,708	42,653	579	1.7	22,708	22,910	1.1
miniDAFJS03	18,363	10,016	153	0.9	18,363	14,385	0.3
miniDAFJS04	20,498	4,002	133	0.3	20,498	5,579	0.2
miniDAFJS05	20,593	44,305	555	2.7	20,593	66,626	3.0
miniDAFJS06	22,867	180,943	1,070	12.0	22,867	80,813	5.5
miniDAFJS07	25,715	18,460	197	1.1	25,715	29,313	0.5
miniDAFJS08	19,878	2,120	21	0.1	19,878	3,003	0.0
miniDAFJS09	24,267	225,648	1,247	13.6	24,267	362,021	11.1
miniDAFJS10	20,336	15,969	352	1.3	20,336	18,971	0.9
miniDAFJS11	29,968	599,704	2,896	39.0	29,968	151,381	6.8
miniDAFJS12	18,670	18,022	400	2.0	18,670	37,514	1.3
miniDAFJS13	16,313	2,056	83	0.2	16,313	4,632	0.1
miniDAFJS14	23,140	109,787	994	7.6	23,140	28,584	2.2
miniDAFJS15	21,715	15,342	251	0.8	21,715	66,937	2.3
miniDAFJS16	25,426	183,188	2,074	9.3	25,426	35,825	2.3
miniDAFJS17	20,155	1,930	57	0.5	20,155	20,812	0.5
miniDAFJS18	18,135	26,554	493	1.3	18,135	19,711	0.4
miniDAFJS19	20,945	50,938	518	2.5	20,945	25,219	1.0
miniDAFJS20	21,838	96,605	660	4.9	21,838	32,301	2.0
miniDAFJS21	23,344	4,518,198	19,262	903.1	23,344	10,098,509	247.3
miniDAFJS22	25,923	1,997,133	10,147	630.9	25,923	746,476	27.5
miniDAFJS23	[22,852, 24,038] 4.93%	11,686,580	73,981	3600.0	24,038	16,261,135	379.0
miniDAFJS24	24,579	653,031	2,661	137.0	24,579	692,014	25.1
miniDAFJS25	21,143	677,724	5,764	156.6	21,143	63,828	5.4
miniDAFJS26	21,120	689,121	4,229	167.5	21,120	428,531	13.8
miniDAFJS27	22,050	3,185,201	15,802	664.3	22,050	2,589,763	73.0
miniDAFJS28	22,708	559,433	3,820	62.6	22,708	376,604	12.6
miniDAFJS29	20,278	56,304	407	10.3	20,278	45,095	3.0
miniDAFJS30	23,558	2,493,830	11,954	595.6	23,558	15,254,029	393.1
miniYFJS01	35,046	26,991	188	4.1	35,046	14,319	1.2
miniYFJS02	24,359	13,715	130	1.6	24,359	44,947	1.0
miniYFJS03	47,391	18,760	149	2.0	47,391	157,935	11.2
miniYFJS04	25,394	116,336	935	15.3	25,394	19,814	2.1
miniYFJS05	23,985	31,811	260	5.4	23,985	15,639	1.7
miniYFJS06	29,469	84,139	673	10.1	29,469	67,644	4.3
miniYFJS07	45,705	40,428	529	4.9	45,705	62,037	5.2
miniYFJS08	33,829	114,680	1,115	12.5	33,829	51,730	3.7
miniYFJS09	37,049	138,409	873	13.8	37,049	41,927	4.8
miniYFJS10	27,310	55,973	399	14.4	27,310	40,806	4.9
miniYFJS11	41,300	79,689	610	11.4	41,300	53,650	5.9
miniYFJS12	30,145	385,442	1,258	50.1	30,145	337,290	27.9
miniYFJS13	30,962	124,224	1,000	23.5	30,962	547,241	39.0
miniYFJS14	31,398	154,413	1,533	26.0	31,398	246,142	18.5
miniYFJS15	45,442	94,003	579	16.8	45,442	135,017	11.4
miniYFJS16	33,791	348,128	2,685	49.9	33,791	252,980	22.5
miniYFJS17	42,838	419,682	1,831	58.4	42,838	323,005	21.2
miniYFJS18	28,247	89,973	788	12.9	28,247	26,872	2.0
miniYFJS19	33,601	406,894	2,149	56.8	33,601	411,937	25.6
miniYFJS20	30,837	125,514	667	21.0	30,837	169,871	20.7
miniYFJS21	37,096	983,262	5,014	186.9	37,096	1,617,533	52.9
miniYFJS22	34,282	1,387,123	6,197	345.3	34,282	2,042,029	99.3
miniYFJS23	42,079	10,588,545	35,879	1859.2	42,079	17,113,272	448.5
miniYFJS24	30,905	475,817	2,710	122.2	30,905	697,688	58.3
miniYFJS25	36,170	2,998,188	12,325	575.1	36,170	8,015,071	261.3
miniYFJS26	51,466	459,386	2,098	160.8	51,466	3,051,373	154.1
miniYFJS27	36,719	1,953,628	3,841	394.0	36,719	915,979	91.6
miniYFJS28	34,509	662,474	2,390	157.7	34,509	328,493	32.9
miniYFJS29	39,798	1,762,678	7,366	407.1	39,798	4,576,379	239.8
miniYFJS30	33,974	903,909	4,828	221.4	33,974	271,588	44.0

Table 5: Solutions found and computational cost of solving the small-sized instances with learning rate $\alpha = 0.1$ using CPLEX and CP Optimizer with no warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
miniDAFJS01	21,327	546,297	4,285	31.5	21,327	953,712	51.7
miniDAFJS02	20,635	81,821	640	3.3	20,635	62,089	1.8
miniDAFJS03	17,972	19,340	305	0.9	17,972	22,808	0.5
miniDAFJS04	19,602	7,746	226	0.3	19,602	13,297	0.9
miniDAFJS05	18,803	98,953	1,120	5.1	18,803	151,224	7.9
miniDAFJS06	20,568	450,187	2,639	38.2	20,568	275,011	19.3
miniDAFJS07	24,715	52,190	642	2.0	24,715	24,272	1.5
miniDAFJS08	18,857	3,493	50	0.1	18,857	4,021	0.2
miniDAFJS09	22,660	1,001,850	6,554	70.8	22,660	784,419	40.0
miniDAFJS10	18,823	10,901	169	0.6	18,823	26,604	0.9
miniDAFJS11	27,455	672,082	3,718	48.2	27,455	416,554	27.4
miniDAFJS12	17,874	70,638	1,078	4.0	17,874	90,535	5.0
miniDAFJS13	15,143	6,291	160	0.3	15,143	15,663	0.5
miniDAFJS14	21,817	332,619	2,896	25.1	21,817	81,858	6.5
miniDAFJS15	20,236	69,714	883	5.5	20,236	116,417	6.4
miniDAFJS16	24,114	357,217	3,894	30.4	24,114	124,890	9.9
miniDAFJS17	19,145	19,035	378	0.8	19,145	16,328	0.7
miniDAFJS18	17,270	32,161	673	1.6	17,270	18,067	0.9
miniDAFJS19	19,642	71,718	942	3.6	19,642	54,772	3.0
miniDAFJS20	20,086	190,012	2,064	14.4	20,086	205,967	19.1
miniDAFJS21	21,352	8,846,059	32,459	1695.1	21,352	45,309,269	2118.7
miniDAFJS22	23,852	3,465,482	19,189	692.6	23,852	5,986,205	571.9
miniDAFJS23	[19,423, 23,228] 16.38%	16,816,032	52,962	3600.0	[14,739, 22,390] 34.17%	112,195,507	3600.0
miniDAFJS24	22,521	2,377,420	12,640	424.6	22,521	12,623,368	836.0
miniDAFJS25	19,809	1,886,995	12,894	394.3	19,809	491,120	69.0
miniDAFJS26	19,724	2,518,201	12,034	343.9	19,724	4,903,993	345.1
miniDAFJS27	[18,804, 20,245] 7.12%	18,924,978	90,951	3600.0	20,245	25,178,221	1289.9
miniDAFJS28	20,635	934,303	6,462	106.2	20,635	688,089	75.9
miniDAFJS29	19,201	217,357	1,998	50.7	19,201	231,066	19.8
miniDAFJS30	21,552	15,710,174	43,128	2940.1	[15,207, 21,552] 29.44%	103,436,263	3600.0
miniYFJS01	33,132	48,887	325	7.4	33,132	46,701	5.4
miniYFJS02	23,100	24,831	219	2.6	23,100	29,314	1.8
miniYFJS03	42,896	98,915	1,106	8.9	42,896	543,066	36.7
miniYFJS04	24,485	58,780	370	5.9	24,485	47,787	4.4
miniYFJS05	23,597	104,135	634	14.4	23,597	46,049	4.6
miniYFJS06	28,655	44,226	318	5.3	28,655	166,757	14.7
miniYFJS07	42,239	37,239	321	3.3	42,239	79,123	7.5
miniYFJS08	31,471	60,079	724	6.1	31,471	120,482	11.0
miniYFJS09	35,250	182,095	1,770	22.8	35,250	129,652	12.5
miniYFJS10	26,145	190,450	1,068	28.0	26,145	230,577	26.2
miniYFJS11	38,545	374,613	1,492	47.9	38,545	103,811	12.1
miniYFJS12	27,895	1,092,118	5,688	230.2	27,895	1,112,993	86.2
miniYFJS13	28,120	401,361	1,771	102.8	28,120	878,304	85.1
miniYFJS14	29,682	598,499	4,869	83.6	29,682	652,854	51.0
miniYFJS15	41,619	55,926	407	9.2	41,619	335,750	31.5
miniYFJS16	31,280	541,733	1,770	83.6	31,280	465,765	57.3
miniYFJS17	40,388	323,307	1,758	76.1	40,388	618,155	51.3
miniYFJS18	26,297	388,013	2,147	37.3	26,297	92,394	8.3
miniYFJS19	30,717	1,063,227	3,558	179.5	30,717	536,931	61.1
miniYFJS20	28,832	579,799	3,548	233.2	28,832	302,287	52.9
miniYFJS21	34,811	5,105,642	17,961	902.4	34,811	3,302,342	307.5
miniYFJS22	31,702	2,121,952	10,758	786.7	31,702	33,812,020	1463.8
miniYFJS23	[34,163, 38,639] 11.58%	19,942,344	52,670	3600.0	[23,930, 38,639] 38.07%	117,925,287	3600.0
miniYFJS24	28,884	739,079	2,875	195.4	28,884	2,439,716	193.4
miniYFJS25	[31,325, 34,231] 8.49%	21,007,500	75,116	3600.0	34,231	45,932,610	1559.7
miniYFJS26	47,519	3,021,164	10,804	646.7	47,519	5,662,793	383.2
miniYFJS27	[33,984, 34,042] 0.17%	10,487,220	76,842	3600.0	34,042	1,256,566	232.8
miniYFJS28	32,080	1,009,471	3,669	268.1	32,080	798,521	109.2
miniYFJS29	36,093	5,412,055	20,761	1095.4	36,093	101,550,404	3074.0
miniYFJS30	31,888	2,077,134	5,979	530.3	31,888	8,395,618	641.4

Table 6: Solutions found and computational cost of solving the small-sized instances with learning rate $\alpha = 0.2$ using CPLEX and CP Optimizer with no warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
miniDAFJS01	19,443	2,072,668	16,713	114.7	19,443	2,800,465	77.8
miniDAFJS02	18,916	53,092	696	2.5	18,916	59,859	2.2
miniDAFJS03	17,419	17,807	254	1.1	17,419	36,788	0.9
miniDAFJS04	18,800	5,444	141	0.4	18,800	19,010	0.7
miniDAFJS05	17,596	149,857	1,357	26.4	17,596	354,775	19.2
miniDAFJS06	18,692	1,036,732	8,974	91.7	18,692	1,030,013	66.3
miniDAFJS07	24,256	223,460	2,563	12.5	24,256	44,462	1.6
miniDAFJS08	17,900	2,439	36	0.2	17,900	7,001	0.1
miniDAFJS09	20,797	1,352,796	6,759	102.3	20,797	5,140,894	148.1
miniDAFJS10	17,395	14,964	318	0.9	17,395	56,602	3.0
miniDAFJS11	25,304	1,300,871	6,954	89.5	25,304	830,064	56.3
miniDAFJS12	17,105	102,571	2,392	6.4	17,105	193,005	9.0
miniDAFJS13	14,077	4,468	83	0.4	14,077	11,633	0.5
miniDAFJS14	20,620	274,434	2,260	33.2	20,620	174,644	11.7
miniDAFJS15	18,625	108,476	1,319	4.5	18,625	231,276	9.8
miniDAFJS16	22,734	688,457	4,387	51.0	22,734	358,341	24.1
miniDAFJS17	18,253	63,437	1,442	2.3	18,253	27,487	1.1
miniDAFJS18	16,495	69,463	1,219	3.1	16,495	24,166	0.8
miniDAFJS19	18,474	50,222	395	2.7	18,474	40,228	2.2
miniDAFJS20	18,521	366,551	3,183	30.0	18,521	448,174	21.7
miniDAFJS21	[16,580, 19,473] 14.86%	23,716,918	114,428	3600.0	[11,665, 19,430] 39.96%	121,044,033	3600.0
miniDAFJS22	22,322	15,610,815	69,887	2530.2	22,322	31,598,705	896.9
miniDAFJS23	[18,647, 20,932] 10.92%	18,939,366	75,870	3600.0	[11,409, 21,031] 45.75%	126,864,753	3600.0
miniDAFJS24	20,389	7,740,688	34,546	997.4	20,389	74,334,372	2170.9
miniDAFJS25	18,400	4,081,716	27,290	695.1	18,400	3,375,974	121.3
miniDAFJS26	18,396	12,778,269	84,472	1781.5	18,396	35,270,716	833.3
miniDAFJS27	18,501	15,343,283	50,257	2684.3	[10,545, 18,501] 43.00%	125,141,223	3600.0
miniDAFJS28	18,762	4,953,263	28,751	430.6	18,762	2,597,465	87.9
miniDAFJS29	18,253	339,168	2,281	53.3	18,253	926,251	58.8
miniDAFJS30	[16,858, 20,137] 16.28%	24,986,837	64,214	3600.0	[11,670, 19,504] 40.17%	119,413,749	3600.0
miniYFJS01	31,008	144,169	1,697	19.6	31,008	58,142	6.7
miniYFJS02	22,010	47,592	623	5.8	22,010	58,225	3.7
miniYFJS03	38,935	48,719	350	6.4	38,935	789,672	57.0
miniYFJS04	23,774	125,088	2,363	25.0	23,774	57,239	7.0
miniYFJS05	22,843	166,373	2,320	21.9	22,843	49,347	6.3
miniYFJS06	27,366	92,146	965	11.1	27,366	222,286	18.5
miniYFJS07	38,932	37,562	391	4.2	38,932	119,422	13.0
miniYFJS08	29,464	189,028	2,554	29.8	29,464	161,280	15.7
miniYFJS09	33,763	478,771	3,515	68.6	33,763	352,444	36.2
miniYFJS10	25,072	544,968	2,549	85.2	25,072	593,197	69.8
miniYFJS11	36,307	318,588	2,654	51.5	36,307	172,185	16.4
miniYFJS12	26,219	887,339	4,393	402.1	26,219	1,005,626	194.1
miniYFJS13	25,619	156,044	1,210	37.1	25,619	5,355,875	539.7
miniYFJS14	27,428	765,456	3,252	112.1	27,428	1,890,640	143.6
miniYFJS15	38,256	92,806	556	12.8	38,256	720,998	79.9
miniYFJS16	29,442	841,644	3,226	235.4	29,442	948,799	183.4
miniYFJS17	37,465	688,837	4,042	179.9	37,465	2,123,022	203.6
miniYFJS18	25,067	472,503	2,385	62.8	25,067	156,755	15.8
miniYFJS19	29,207	1,139,515	3,644	187.3	29,207	1,685,954	190.8
miniYFJS20	27,091	726,398	3,982	288.1	27,091	2,476,745	323.9
miniYFJS21	[31,599, 32,166] 1.76%	20,680,874	61,929	3600.0	32,166	13,121,393	1009.8
miniYFJS22	28,985	7,719,748	18,201	2445.2	[17,383, 28,985] 40.03%	81,047,313	3600.0
miniYFJS23	[28,463, 37,117] 23.32%	17,784,252	31,743	3600.0	[18,601, 35,441] 47.52%	102,566,851	3600.0
miniYFJS24	27,023	1,614,894	6,496	358.3	27,023	1,258,664	175.7
miniYFJS25	32,346	13,267,205	60,508	2580.0	[17,438, 32,346] 46.09%	112,250,946	3600.0
miniYFJS26	43,452	835,404	4,571	299.4	43,452	17,463,308	1453.5
miniYFJS27	31,571	1,946,497	6,451	685.6	31,571	1,567,754	398.8
miniYFJS28	30,428	2,509,256	6,433	490.1	30,428	1,762,797	262.3
miniYFJS29	32,826	9,429,478	47,201	2933.9	[17,217, 32,826] 47.55%	72,931,315	3600.0
miniYFJS30	[27,281, 30,427] 10.34%	14,301,672	40,562	3600.0	29,848	50,040,117	1840.0

Table 7: Solutions found and computational cost of solving the small-sized instances with learning rate $\alpha = 0.3$ using CPLEX and CP Optimizer with no warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
DAFJS01	[20,320, 28,850] 29.57%	2,781,484	10,131	3,600.0	[18,617, 24,424] 23.78%	8,687,672	3,600.0
DAFJS02	[23,614, 31,946] 26.08%	2,850,568	10,711	3,600.0	[18,533, 26,575] 30.26%	11,632,154	3,600.0
DAFJS03				3,600.0	[41,271, 54,098] 23.71%	801,461	3,600.1
DAFJS04	[45,286, 131,061] 65.45%	215,396	13	3,600.0	[44,062, 53,100] 17.02%	850,903	3,600.2
DAFJS05	[30,405, 140,075] 78.29%	291,690	510	3,600.0	[23,378, 36,347] 35.68%	2,711,057	3,600.1
DAFJS06	[31,881, 147,509] 78.39%	259,758	320	3,600.0	[23,364, 37,043] 36.93%	1,783,500	3,600.1
DAFJS07				3,600.0	[33,601, 53,504] 37.20%	244,414	3,600.1
DAFJS08				3,600.0	[43,331, 56,281] 23.01%	269,955	3,600.7
DAFJS09	[36,374, 192,570] 81.11%	175,545	72	3,600.0	[22,460, 41,791] 46.26%	965,188	3,600.5
DAFJS10	[39,597, 191,176] 79.29%	98,599	10	3,600.0	[23,440, 49,083] 52.24%	840,177	3,600.1
DAFJS11				3,600.0	[44,061, 75,805] 41.88%	105,184	3,600.6
DAFJS12				3,600.0	[35,262, 81,972] 56.98%	95,935	3,601.9
DAFJS13	[50,247, 244,658] 79.46%	61,459	38	3,600.0	[20,981, 59,747] 64.88%	554,985	3,600.8
DAFJS14	[55,319, 249,678] 77.84%	67,950	35	3,600.0	[23,996, 67,444] 64.42%	520,032	3,600.1
DAFJS15				3,600.0	[33,814, 87,500] 61.36%	99,363	3,601.1
DAFJS16				3,600.0	[42,465, 80,415] 47.19%	102,647	3,601.4
DAFJS17	[57,432, 280,020] 79.49%	67,245	0	3,600.0	[20,260, 71,425] 71.63%	365,553	3,600.5
DAFJS18	[58,268, 297,443] 80.41%	48,679	12	3,600.0	[21,838, 76,769] 71.55%	339,613	3,600.3
DAFJS19	[36,126, 303,143] 88.08%	49,902	5	3,600.0	[35,267, 52,861] 33.28%	345,133	3,601.0
DAFJS20				3,600.0	[29,175, 73,693] 60.41%	252,777	3,600.5
DAFJS21				3,600.0	[33,405, 84,030] 60.25%	179,517	3,600.9
DAFJS22				3,600.0	[30,309, 86,845] 65.10%	135,315	3,602.9
DAFJS23				3,600.0	[31,047, 49,850] 37.72%	433,417	3,601.8
DAFJS24				3,600.0	[32,194, 65,845] 51.11%	208,687	3,600.6
DAFJS25				3,600.0	[38,155, 97,200] 60.75%	94,874	3,601.5
DAFJS26				3,600.0	[36,963, 88,141] 58.06%	127,700	3,602.7
DAFJS27				3,600.0	[32,947, 101,100] 67.41%	100,670	3,602.2
DAFJS28				3,600.0	[36,392, 57,672] 36.90%	197,677	3,600.6
DAFJS29				3,600.0	[40,983, 65,385] 37.32%	204,841	3,600.7
DAFJS30				3,600.0	[31,515, 63,194] 50.13%	217,698	3,600.8
YFJS01	[57,696, 231,813] 75.11%	1,238,787	3,343	3,600.0	[53,549, 69,362] 22.80%	7,616,521	3,600.0
YFJS02	[65,884, 254,446] 74.11%	3,832,008	7,555	3,600.0	[61,356, 72,465] 15.33%	17,752,225	3,600.0
YFJS03	32,538	2,006,700	10,709	541.2	32,538	2,940,455	168.8
YFJS04	35,883	5,283,854	24,497	1,843.3	[27,197, 35,883] 24.21%	23,523,339	3,600.0
YFJS05	[37,902, 44,039] 13.94%	5,926,501	22,100	3,600.0	[36,307, 40,186] 9.65%	34,964,540	3,600.0
YFJS06	[35,243, 51,826] 32.00%	2,722,415	10,769	3,600.0	[25,961, 41,178] 36.95%	16,247,718	3,600.0
YFJS07	[36,346, 104,736] 65.30%	2,698,037	5,692	3,600.0	[32,925, 41,534] 20.73%	18,901,026	3,600.0
YFJS08	32,573	3,051,228	7,177	708.6	32,573	1,259,189	216.5
YFJS09	[17,755, 40,126] 55.75%	627,852	1,029	3,600.0	[15,238, 22,745] 33.01%	4,864,764	3,600.0
YFJS10	[31,850, 40,178] 20.73%	8,068,466	27,728	3,600.0	[33,897, 37,372] 9.30%	28,533,789	3,600.0
YFJS11	[42,604, 132,168] 67.77%	848,614	1,063	3,600.0	[39,044, 49,425] 21.00%	10,127,040	3,600.0
YFJS12	[38,233, 100,230] 61.85%	985,456	1,407	3,600.0	[33,937, 47,572] 28.66%	9,207,619	3,600.0
YFJS13	[30,132, 238,928] 87.39%	909,917	1,608	3,600.0	[27,343, 37,361] 26.81%	7,593,992	3,600.0
YFJS14				3,600.0	[94,991, 117,280] 19.00%	1,036,376	3,601.7
YFJS15				3,600.0	[89,818, 112,561] 20.21%	812,403	3,600.8
YFJS16				3,600.0	[86,618, 121,962] 28.98%	919,286	3,601.0
YFJS17				3,600.0	[76,416, 235,919] 67.61%	103,006	3,600.7
YFJS18				3,600.0	[81,982, 242,241] 66.16%	164,161	3,602.0
YFJS19				3,600.0	[62,129, 222,300] 72.05%	182,566	3,601.4
YFJS20				3,600.0	[65,048, 204,465] 68.19%	209,968	3,602.5

Table 8: Solutions found and computational cost of solving the large-sized instances with learning rate $\alpha = 0.1$ using CPLEX and CP Optimizer with no warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
DAFJS01	[17,243, 24,221] 28.81%	4,060,290	15,216	3,600.0	[14,141, 21,955] 35.59%	10,060,921	3,600.1
DAFJS02	[20,460, 29,199] 29.93%	4,007,884	12,017	3,600.0	[13,966, 24,524] 43.05%	9,597,884	3,600.0
DAFJS03				3,600.0	[29,519, 46,041] 35.89%	805,535	3,600.2
DAFJS04	[33,806, 120,313] 71.90%	227,790	80	3,600.0	[32,038, 46,931] 31.73%	941,659	3,600.4
DAFJS05	[25,343, 121,192] 79.09%	492,382	517	3,600.0	[17,025, 32,708] 47.95%	2,914,026	3,600.1
DAFJS06	[25,463, 140,367] 81.86%	167,625	52	3,600.0	[16,745, 32,824] 48.99%	1,301,726	3,600.2
DAFJS07				3,600.0	[23,042, 48,878] 52.86%	225,447	3,600.4
DAFJS08				3,600.0	[29,898, 50,257] 40.51%	313,129	3,600.3
DAFJS09	[28,979, 145,250] 80.05%	228,678	164	3,600.0	[16,016, 37,560] 57.36%	1,098,208	3,600.8
DAFJS10	[30,054, 156,114] 80.75%	119,791	32	3,600.0	[16,354, 38,870] 57.93%	728,566	3,600.1
DAFJS11				3,600.0	[29,499, 81,300] 63.72%	99,427	3,600.7
DAFJS12				3,600.0	[23,461, 72,400] 67.60%	135,347	3,601.1
DAFJS13	[39,222, 196,510] 80.04%	74,615	68	3,600.0	[14,433, 51,221] 71.82%	453,040	3,600.3
DAFJS14	[43,933, 198,907] 77.91%	45,790	23	3,600.0	[16,518, 57,141] 71.09%	524,897	3,600.2
DAFJS15				3,600.0	[22,324, 79,631] 71.97%	130,605	3,601.1
DAFJS16				3,600.0	[28,167, 81,701] 65.52%	108,595	3,601.6
DAFJS17	[42,564, 220,059] 80.66%	33,743	9	3,600.0	[13,683, 82,646] 83.44%	325,206	3,600.3
DAFJS18	[45,690, 235,050] 80.56%	25,576	24	3,600.0	[14,813, 62,738] 76.39%	336,152	3,600.3
DAFJS19	[27,775, 240,355] 88.44%	48,308	28	3,600.0	[24,250, 44,680] 45.73%	327,636	3,601.5
DAFJS20				3,600.0	[19,568, 63,643] 69.25%	210,906	3,600.7
DAFJS21				3,600.0	[22,142, 87,819] 74.79%	179,311	3,600.7
DAFJS22				3,600.0	[19,800, 85,714] 76.90%	126,624	3,601.2
DAFJS23	[23,925, 233,720] 89.76%	41,051	6	3,600.0	[21,422, 43,475] 50.73%	336,352	3,600.4
DAFJS24				3,600.0	[21,763, 48,898] 55.49%	282,253	3,601.7
DAFJS25				3,600.0	[24,843, 88,803] 72.02%	94,333	3,601.6
DAFJS26				3,600.0	[24,182, 92,386] 73.83%	106,909	3,601.8
DAFJS27				3,600.0	[21,584, 104,021] 79.25%	122,926	3,601.1
DAFJS28				3,600.0	[24,712, 51,123] 51.66%	410,247	3,600.5
DAFJS29				3,600.0	[27,497, 63,976] 57.02%	245,480	3,600.8
DAFJS30				3,600.0	[21,222, 52,467] 59.55%	148,359	3,600.9
YFJS01	[48,380, 207,840] 76.72%	1,396,979	3,117	3,600.0	[39,902, 64,326] 37.97%	8,194,144	3,600.0
YFJS02	[56,014, 110,194] 49.17%	4,003,670	12,006	3,600.0	[46,660, 66,853] 30.21%	16,482,400	3,600.0
YFJS03	30,073	1,963,718	8,908	417.1	30,073	1,773,318	72.1
YFJS04	[28,652, 33,302] 13.96%	9,950,004	30,017	3,600.0	[21,857, 32,670] 33.10%	38,796,057	3,600.0
YFJS05	[34,538, 41,872] 17.51%	5,831,217	22,402	3,600.0	[24,921, 38,628] 35.48%	25,567,988	3,600.0
YFJS06	[30,840, 43,441] 29.01%	3,078,999	12,300	3,600.0	[19,541, 37,181] 47.44%	15,078,425	3,600.0
YFJS07	[33,082, 44,007] 24.83%	3,617,331	11,662	3,600.0	[26,063, 37,758] 30.97%	19,115,592	3,600.0
YFJS08	30,184	9,794,233	20,272	2,876.9	30,184	7,683,812	833.2
YFJS09	[15,117, 158,694] 90.47%	617,130	1,487	3,600.0	[11,324, 21,796] 48.05%	3,596,887	3,600.1
YFJS10	[26,194, 34,624] 24.35%	8,728,009	20,709	3,600.0	[22,368, 34,689] 35.52%	29,087,297	3,600.0
YFJS11	[35,267, 253,872] 86.11%	876,414	650	3,600.0	[30,073, 44,294] 32.11%	11,082,859	3,600.0
YFJS12	[31,237, 252,722] 87.64%	931,005	1,435	3,600.0	[26,939, 43,890] 38.62%	5,091,417	3,600.0
YFJS13	[24,896, 134,313] 81.46%	910,968	1,009	3,600.0	[21,361, 34,961] 38.90%	10,804,651	3,600.0
YFJS14				3,600.0	[68,423, 112,188] 39.01%	1,166,881	3,600.3
YFJS15				3,600.0	[64,923, 108,629] 40.23%	1,190,623	3,600.2
YFJS16				3,600.0	[62,865, 100,740] 37.60%	893,979	3,601.3
YFJS17				3,600.0	[51,540, 227,121] 77.31%	194,201	3,604.5
YFJS18				3,600.0	[55,101, 244,159] 77.43%	206,239	3,601.5
YFJS19				3,600.0	[41,690, 220,400] 81.08%	162,093	3,601.2
YFJS20				3,600.0	[43,711, 221,741] 80.29%	164,334	3,602.6

Table 9: Solutions found and computational cost of solving the large-sized instances with learning rate $\alpha = 0.2$ using CPLEX and CP Optimizer with no warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
DAFJS01	[15,541, 24,191] 35.76%	3,223,970	12,860	3,600.0	[10,711, 20,080] 46.66%	7,023,413	3,600.1
DAFJS02	[16,850, 27,398] 38.50%	3,318,330	12,085	3,600.0	[10,500, 22,335] 52.99%	9,620,254	3,600.0
DAFJS03	[22,536, 165,343] 86.37%	125,755	7	3,600.0	[21,152, 41,224] 48.69%	920,372	3,600.2
DAFJS04	[24,891, 120,636] 79.37%	197,181	48	3,600.0	[23,278, 41,388] 43.76%	785,196	3,600.4
DAFJS05	[20,613, 101,212] 79.63%	482,040	586	3,600.0	[12,401, 28,360] 56.27%	2,582,449	3,600.1
DAFJS06	[20,173, 101,490] 80.12%	155,197	120	3,600.0	[12,004, 29,142] 58.81%	1,249,407	3,600.4
DAFJS07				3,600.0	[15,801, 43,615] 63.77%	268,582	3,601.0
DAFJS08				3,600.0	[20,629, 46,117] 55.27%	344,899	3,600.6
DAFJS09	[22,200, 128,596] 82.74%	273,485	52	3,600.0	[11,423, 32,928] 65.31%	1,100,742	3,600.7
DAFJS10	[22,505, 125,666] 82.09%	91,470	51	3,600.0	[11,408, 34,844] 67.26%	638,065	3,600.1
DAFJS11				3,600.0	[19,709, 68,718] 71.32%	175,014	3,600.8
DAFJS12				3,600.0	[15,608, 74,389] 79.02%	165,507	3,601.0
DAFJS13	[29,196, 162,727] 82.06%	57,780	12	3,600.0	[9,929, 42,863] 76.84%	529,927	3,600.1
DAFJS14	[32,943, 160,464] 79.47%	58,666	28	3,600.0	[11,318, 49,446] 77.11%	446,682	3,600.2
DAFJS15				3,600.0	[14,671, 81,186] 81.93%	243,450	3,601.4
DAFJS16				3,600.0	[18,661, 83,128] 77.55%	92,594	3,601.7
DAFJS17	[29,948, 174,560] 82.84%	28,368	0	3,600.0	[9,240, 53,632] 82.77%	325,179	3,600.5
DAFJS18	[32,363, 188,164] 82.80%	24,901	20	3,600.0	[10,047, 57,057] 82.39%	375,922	3,600.3
DAFJS19	[21,110, 174,022] 87.87%	47,203	24	3,600.0	[16,675, 44,216] 62.29%	344,220	3,600.8
DAFJS20				3,600.0	[13,124, 70,212] 81.31%	214,594	3,601.3
DAFJS21				3,600.0	[14,664, 82,400] 82.20%	153,028	3,601.1
DAFJS22				3,600.0	[12,934, 83,000] 84.42%	206,899	3,601.5
DAFJS23				3,600.0	[14,782, 39,390] 62.47%	321,790	3,600.2
DAFJS24				3,600.0	[14,688, 59,521] 75.32%	199,251	3,600.7
DAFJS25				3,600.0	[16,174, 91,876] 82.40%	104,229	3,601.4
DAFJS26				3,600.0	[15,820, 92,135] 82.83%	161,119	3,601.4
DAFJS27				3,600.0	[14,138, 101,680] 86.10%	99,450	3,601.5
DAFJS28				3,600.0	[16,781, 53,598] 68.69%	296,149	3,600.8
DAFJS29				3,600.0	[18,452, 51,935] 64.47%	189,417	3,600.6
DAFJS30				3,600.0	[14,275, 45,882] 68.89%	376,298	3,600.1
YFJS01	[38,603, 206,193] 81.28%	1,572,862	2,749	3,600.0	[29,808, 57,430] 48.10%	6,949,608	3,600.0
YFJS02	[47,466, 135,328] 64.93%	2,828,906	7,612	3,600.0	[35,526, 59,001] 39.79%	12,658,211	3,600.0
YFJS03	27,686	1,251,537	4,381	296.2	27,686	1,594,601	122.2
YFJS04	[26,169, 32,060] 18.37%	10,641,643	39,250	3,600.0	[18,161, 29,692] 38.84%	27,504,395	3,600.0
YFJS05	[30,039, 36,387] 17.45%	8,525,830	25,348	3,600.0	[19,314, 34,779] 44.47%	22,727,062	3,600.0
YFJS06	[26,525, 48,115] 44.87%	3,358,590	10,847	3,600.0	[14,714, 35,191] 58.19%	14,821,556	3,600.0
YFJS07	[25,837, 38,561] 33.00%	6,699,356	10,466	3,600.0	[20,397, 34,719] 41.25%	19,712,483	3,600.0
YFJS08	28,192	13,666,904	32,817	2,983.1	[17,787, 28,192] 36.91%	28,825,696	3,600.0
YFJS09	[12,140, 58,851] 79.37%	612,060	505	3,600.0	[9,525, 20,298] 53.07%	3,309,202	3,600.0
YFJS10	[26,137, 33,956] 23.03%	8,801,436	30,736	3,600.0	[17,808, 32,159] 44.63%	14,620,248	3,600.0
YFJS11	[29,484, 105,275] 71.99%	1,018,382	2,382	3,600.0	[23,170, 40,504] 42.80%	7,022,214	3,600.0
YFJS12	[25,803, 219,386] 88.24%	800,690	1,458	3,600.0	[21,335, 40,292] 47.05%	8,198,562	3,600.0
YFJS13	[20,982, 157,470] 86.68%	810,584	1,229	3,600.0	[16,668, 31,626] 47.30%	8,516,433	3,600.0
YFJS14				3,600.0	[49,252, 94,421] 47.84%	969,495	3,600.2
YFJS15				3,600.0	[46,947, 94,408] 50.27%	973,106	3,600.2
YFJS16				3,600.0	[45,648, 94,031] 51.45%	944,283	3,600.2
YFJS17				3,600.0	[34,763, 216,587] 83.95%	202,741	3,602.6
YFJS18				3,600.0	[37,042, 246,800] 84.99%	138,109	3,601.5
YFJS19				3,600.0	[27,976, 199,757] 85.99%	169,481	3,601.1
YFJS20				3,600.0	[29,376, 108,103] 72.83%	1,201,418	3,600.8

Table 10: Solutions found and computational cost of solving the large-sized instances with learning rate $\alpha = 0.3$ using CPLEX and CP Optimizer with no warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
miniDAFJS01	22,875	27,130	272	2.2	22,875	143,152	3.2
miniDAFJS02	22,708	9,960	85	1.1	22,708	15,317	0.3
miniDAFJS03	18,363	0	0	0.0	18,363	5,403	0.1
miniDAFJS04	20,498	155	0	0.0	20,498	27,577	0.2
miniDAFJS05	20,593	27,452	195	1.7	20,593	66,727	1.1
miniDAFJS06	22,867	4,996	28	1.1	22,867	67,777	1.6
miniDAFJS07	25,715	368	3	0.1	25,715	21,890	0.3
miniDAFJS08	19,878	0	0	-	19,878	2,920	0.0
miniDAFJS09	24,267	81,679	379	5.4	24,267	216,182	4.9
miniDAFJS10	20,336	725	0	0.1	20,336	22,466	0.2
miniDAFJS11	29,968	199,078	963	9.8	29,968	175,679	3.2
miniDAFJS12	18,670	2,687	48	0.4	18,670	30,780	1.0
miniDAFJS13	16,313	225	0	0.0	16,313	3,313	0.1
miniDAFJS14	23,140	8,076	76	0.7	23,140	22,300	0.5
miniDAFJS15	21,715	5,253	86	0.4	21,715	41,419	0.6
miniDAFJS16	25,426	4,790	40	1.4	25,426	19,747	1.3
miniDAFJS17	20,155	126	0	0.0	20,155	13,033	0.2
miniDAFJS18	18,135	525	0	0.1	18,135	36,894	0.3
miniDAFJS19	20,945	54	0	0.1	20,945	13,158	0.5
miniDAFJS20	21,838	9,612	81	0.7	21,838	102,174	1.5
miniDAFJS21	23,344	2,836,539	11,309	506.7	23,344	14,223,831	668.3
miniDAFJS22	25,923	1,458,988	5,824	400.1	25,923	1,677,181	145.2
miniDAFJS23	24,038	11,310,339	53,964	2,311.6	24,038	21,223,801	1,037.8
miniDAFJS24	24,579	466,199	2,231	84.8	24,579	1,360,319	54.6
miniDAFJS25	21,143	205,001	1,097	32.3	21,143	122,087	9.4
miniDAFJS26	21,120	678,221	3,053	160.1	21,120	384,449	30.3
miniDAFJS27	22,050	4,215,023	21,045	660.6	22,050	1,437,370	107.7
miniDAFJS28	22,708	1,861,961	13,443	186.5	22,708	275,433	12.9
miniDAFJS29	20,278	7,016	36	1.3	20,278	34,767	2.5
miniDAFJS30	23,558	8,456,954	23,560	1,500.5	23,558	15,938,770	541.0
miniYFJS01	35,046	103	0	0.1	35,046	10,467	0.8
miniYFJS02	24,359	3,241	50	0.4	24,359	21,396	0.7
miniYFJS03	47,391	18,165	76	2.6	47,391	127,728	7.5
miniYFJS04	25,394	0	0	0.1	25,394	11,912	0.7
miniYFJS05	23,985	6,849	149	1.0	23,985	29,709	2.5
miniYFJS06	29,469	6,924	85	1.0	29,469	53,903	4.0
miniYFJS07	45,705	9,538	46	1.6	45,705	76,457	4.3
miniYFJS08	33,829	25,821	97	3.2	33,829	51,912	3.9
miniYFJS09	37,049	18,799	72	2.2	37,049	38,103	2.9
miniYFJS10	27,310	57,031	322	7.8	27,310	82,248	6.9
miniYFJS11	41,300	12,019	39	2.3	41,300	41,763	2.9
miniYFJS12	30,145	294,626	2,569	77.5	30,145	331,536	28.0
miniYFJS13	30,962	102,796	961	19.0	30,962	901,505	38.0
miniYFJS14	31,398	183,367	1,750	16.5	31,398	236,377	13.0
miniYFJS15	45,442	82,620	330	8.2	45,442	105,976	7.0
miniYFJS16	33,791	54,694	430	15.4	33,791	174,917	14.3
miniYFJS17	42,838	179,623	955	25.0	42,838	282,364	20.0
miniYFJS18	28,247	42,054	291	7.2	28,247	56,100	3.3
miniYFJS19	33,601	290,798	2,054	26.7	33,601	346,272	23.0
miniYFJS20	30,837	130,053	955	21.0	30,837	107,625	12.6
miniYFJS21	37,096	739,545	2,458	116.2	37,096	1,094,577	37.3
miniYFJS22	34,282	383,245	2,448	88.7	34,282	1,202,777	56.8
miniYFJS23	42,079	1,273,019	3,794	211.1	42,079	11,638,509	300.8
miniYFJS24	30,905	508,302	2,030	80.6	30,905	502,409	42.8
miniYFJS25	36,170	1,461,404	6,280	274.0	36,170	9,574,183	267.5
miniYFJS26	51,466	180,881	1,916	38.9	51,466	2,962,339	111.9
miniYFJS27	36,719	1,046,266	4,945	269.3	36,719	370,959	53.3
miniYFJS28	34,509	355,782	2,272	75.0	34,509	309,959	29.5
miniYFJS29	39,798	924,452	3,149	201.2	39,798	5,971,499	217.6
miniYFJS30	33,974	984,016	4,217	229.0	33,974	398,470	52.5

Table 11: Solutions found and computational cost of solving the small-sized instances with learning rate $\alpha = 0.1$ using CPLEX and CP Optimizer with warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
miniDAFJS01	21,327	539,475	4,331	31.9	21,327	759,456	13.6
miniDAFJS02	20,635	30,346	207	2.0	20,635	30,019	0.5
miniDAFJS03	17,972	561	0	0.1	17,972	10,221	0.3
miniDAFJS04	19,602	655	0	0.1	19,602	14,122	0.2
miniDAFJS05	18,803	58,235	514	3.5	18,803	206,620	4.5
miniDAFJS06	20,568	100,122	455	7.1	20,568	546,000	9.3
miniDAFJS07	24,715	949	0	0.1	24,715	18,941	0.3
miniDAFJS08	18,857	0	0	-	18,857	1,649	0.0
miniDAFJS09	22,660	186,037	926	11.0	22,660	883,945	18.5
miniDAFJS10	18,823	5,108	109	0.4	18,823	26,280	0.3
miniDAFJS11	27,455	300,137	1,078	16.7	27,455	374,425	7.7
miniDAFJS12	17,874	12,831	131	1.2	17,874	59,342	1.3
miniDAFJS13	15,143	2,141	55	0.2	15,143	19,480	0.2
miniDAFJS14	21,817	70,714	493	4.4	21,817	74,338	1.5
miniDAFJS15	20,236	37,460	459	2.0	20,236	87,133	1.7
miniDAFJS16	24,114	64,350	482	3.6	24,114	307,748	6.8
miniDAFJS17	19,145	829	0	0.2	19,145	15,518	0.3
miniDAFJS18	17,270	1,065	0	0.2	17,270	17,709	0.5
miniDAFJS19	19,642	1,547	0	0.4	19,642	21,831	0.6
miniDAFJS20	20,086	29,709	216	1.9	20,086	153,453	2.4
miniDAFJS21	21,352	18,397,933	64,892	3,232.2	21,352	48,115,711	1,848.5
miniDAFJS22	23,852	2,278,871	8,970	750.5	23,852	5,245,403	434.6
miniDAFJS23	[21,765, 22,491] 3.23%	18,863,739	62,979	3,600.0	[14,739, 22,390] 34.17%	111,132,643	3,600.0
miniDAFJS24	22,521	867,832	3,978	139.4	22,521	8,980,581	264.8
miniDAFJS25	19,809	1,379,354	6,519	212.9	19,809	562,324	37.1
miniDAFJS26	19,724	2,047,373	8,877	233.3	19,724	2,193,339	95.1
miniDAFJS27	20,245	20,712,585	48,282	2,813.0	20,245	50,918,190	1,460.9
miniDAFJS28	20,635	1,901,205	7,073	282.7	20,635	469,515	14.7
miniDAFJS29	19,201	31,346	132	7.3	19,201	324,996	8.7
miniDAFJS30	21,552	7,330,089	24,386	1,396.9	21,552	121,294,838	3,525.5
miniYFJS01	33,132	3,140	10	0.7	33,132	36,116	3.5
miniYFJS02	23,100	7,994	68	1.1	23,100	29,477	1.7
miniYFJS03	42,896	30,087	187	3.4	42,896	778,560	42.4
miniYFJS04	24,485	1,988	0	0.4	24,485	34,244	3.7
miniYFJS05	23,597	6,272	130	0.8	23,597	28,855	3.1
miniYFJS06	28,655	14,327	71	1.8	28,655	96,238	7.6
miniYFJS07	42,239	13,236	256	2.0	42,239	67,142	6.4
miniYFJS08	31,471	105,081	743	17.0	31,471	104,921	6.3
miniYFJS09	35,250	57,100	909	8.8	35,250	127,601	8.7
miniYFJS10	26,145	77,158	308	11.6	26,145	327,857	29.9
miniYFJS11	38,545	24,787	119	4.2	38,545	82,328	8.6
miniYFJS12	27,895	553,149	2,713	122.3	27,895	558,847	50.6
miniYFJS13	28,120	66,786	456	17.8	28,120	764,195	60.8
miniYFJS14	29,682	430,434	3,103	44.7	29,682	627,089	43.5
miniYFJS15	41,619	59,634	221	8.6	41,619	396,413	28.1
miniYFJS16	31,280	93,638	810	20.9	31,280	444,883	44.2
miniYFJS17	40,388	327,916	2,106	48.6	40,388	770,351	43.9
miniYFJS18	26,297	88,571	326	9.0	26,297	105,187	9.5
miniYFJS19	30,717	835,718	3,653	243.0	30,717	471,674	53.1
miniYFJS20	28,832	370,354	2,191	120.3	28,832	703,276	88.2
miniYFJS21	34,811	6,145,627	21,227	977.9	34,811	5,002,760	274.9
miniYFJS22	31,702	1,643,715	5,521	642.9	31,702	23,565,438	1,002.5
miniYFJS23	38,639	17,107,906	40,129	2,673.9	[23,930, 38,639] 38.07%	134,588,957	3,600.0
miniYFJS24	28,884	739,701	2,619	162.6	28,884	1,758,952	124.2
miniYFJS25	[31,852, 34,231] 6.95%	24,701,462	59,283	3,600.0	34,231	43,322,143	1,291.2
miniYFJS26	47,519	465,569	3,359	180.3	47,519	6,832,013	361.0
miniYFJS27	34,042	519,541	1,954	209.2	34,042	1,960,966	199.3
miniYFJS28	32,080	413,378	1,225	73.8	32,080	710,691	71.6
miniYFJS29	36,093	1,214,346	3,669	333.5	36,093	70,667,722	2,208.3
miniYFJS30	31,888	3,633,325	10,645	763.9	31,888	10,290,751	500.6

Table 12: Solutions found and computational cost of solving the small-sized instances with learning rate $\alpha = 0.2$ using CPLEX and CP Optimizer with warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
miniDAFJS01	19,443	698,096	4,945	38.9	19,443	1,528,271	32.0
miniDAFJS02	18,916	43,060	324	2.0	18,916	81,435	1.5
miniDAFJS03	17,419	6,368	91	0.6	17,419	27,973	0.4
miniDAFJS04	18,800	4,986	135	0.3	18,800	23,757	0.3
miniDAFJS05	17,596	174,306	2,110	7.9	17,596	505,368	9.0
miniDAFJS06	18,692	443,238	2,499	22.7	18,692	1,037,811	22.1
miniDAFJS07	24,256	13,304	243	0.8	24,256	53,533	0.9
miniDAFJS08	17,900	111	0	0.0	17,900	4,197	0.1
miniDAFJS09	20,797	414,070	1,767	22.8	20,797	3,702,199	71.1
miniDAFJS10	17,395	4,313	63	0.5	17,395	39,582	0.6
miniDAFJS11	25,304	879,945	4,672	52.5	25,304	652,569	13.6
miniDAFJS12	17,105	44,853	368	2.5	17,105	146,544	2.9
miniDAFJS13	14,077	1,996	31	0.2	14,077	18,277	0.3
miniDAFJS14	20,620	228,333	1,904	18.2	20,620	290,223	5.1
miniDAFJS15	18,625	43,116	709	3.2	18,625	254,148	4.0
miniDAFJS16	22,734	192,603	1,353	8.9	22,734	397,403	8.5
miniDAFJS17	18,253	4,414	116	0.3	18,253	44,549	0.6
miniDAFJS18	16,495	18,129	299	1.1	16,495	31,340	0.7
miniDAFJS19	18,474	8,267	49	0.9	18,474	62,253	1.2
miniDAFJS20	18,521	76,034	409	5.9	18,521	570,319	27.0
miniDAFJS21	19,430	12,431,780	94,510	2,459.4	[11,665, 19,430] 39.96%	105,058,989	3,600.0
miniDAFJS22	22,322	3,599,003	16,569	682.7	22,322	25,436,965	1,435.7
miniDAFJS23	[18,434, 20,932] 11.94%	18,976,637	71,111	3,600.0	[11,409, 20,932] 45.49%	95,215,268	3,600.0
miniDAFJS24	20,389	5,941,541	24,041	704.4	20,389	83,999,123	2,067.1
miniDAFJS25	18,400	1,455,883	5,536	238.6	18,400	2,232,839	148.1
miniDAFJS26	18,396	3,666,483	30,795	533.9	18,396	31,939,819	783.1
miniDAFJS27	18,501	24,132,489	84,513	3,505.6	[10,545, 18,501] 43.00%	122,850,832	3,600.0
miniDAFJS28	18,762	4,874,294	33,717	418.7	18,762	2,637,043	142.9
miniDAFJS29	18,253	32,091	183	5.0	18,253	629,042	23.9
miniDAFJS30	[17,441, 19,618] 11.10%	27,578,433	109,651	3,600.0	[11,670, 19,504] 40.17%	129,675,062	3,600.0
miniYFJS01	31,008	10,554	42	2.3	31,008	58,728	7.3
miniYFJS02	22,010	12,966	86	1.7	22,010	40,643	2.8
miniYFJS03	38,935	36,885	205	4.4	38,935	973,233	54.5
miniYFJS04	23,774	3,571	20	0.6	23,774	62,640	7.0
miniYFJS05	22,843	19,695	119	3.1	22,843	74,641	8.0
miniYFJS06	27,366	20,266	49	2.6	27,366	141,515	13.2
miniYFJS07	38,932	8,969	120	2.0	38,932	152,186	12.2
miniYFJS08	29,464	105,069	606	8.6	29,464	164,477	13.5
miniYFJS09	33,763	228,616	1,229	17.7	33,763	436,472	32.1
miniYFJS10	25,072	330,870	2,304	53.3	25,072	722,733	56.6
miniYFJS11	36,307	47,377	224	9.7	36,307	132,322	15.1
miniYFJS12	26,219	1,091,871	4,786	342.8	26,219	1,172,626	196.8
miniYFJS13	25,619	64,547	257	15.2	25,619	4,736,621	403.7
miniYFJS14	27,428	1,341,266	10,416	199.0	27,428	1,441,449	147.9
miniYFJS15	38,256	113,347	531	12.9	38,256	764,677	59.9
miniYFJS16	29,442	129,890	593	31.3	29,442	1,172,679	138.6
miniYFJS17	37,465	831,549	2,535	96.6	37,465	867,735	106.3
miniYFJS18	25,067	109,409	268	11.2	25,067	230,384	19.2
miniYFJS19	29,207	692,645	2,990	127.8	29,207	2,273,168	217.8
miniYFJS20	27,091	273,694	2,124	83.5	27,091	1,673,408	188.3
miniYFJS21	32,166	10,577,650	45,662	2,754.5	32,166	21,292,838	1,363.1
miniYFJS22	28,985	5,415,929	14,497	1,324.2	[17,383, 28,985] 40.03%	109,784,943	3,600.0
miniYFJS23	[29,176, 36,873] 20.88%	20,854,030	82,637	3,600.0	[18,601, 35,441] 47.52%	113,483,666	3,600.0
miniYFJS24	27,023	1,487,508	5,429	343.6	27,023	1,057,202	114.5
miniYFJS25	[31,821, 32,346] 1.62%	22,189,124	79,357	3,600.0	[17,438, 32,465] 46.29%	134,302,757	3,600.0
miniYFJS26	43,452	896,287	3,959	291.0	43,452	14,604,568	663.7
miniYFJS27	31,571	546,712	2,309	367.4	31,571	3,042,229	443.4
miniYFJS28	30,428	919,151	2,475	223.8	30,428	1,426,748	168.6
miniYFJS29	32,826	4,884,746	8,355	1,341.8	[17,217, 32,826] 47.55%	108,180,711	3,600.0
miniYFJS30	29,848	2,521,758	9,078	825.1	29,848	28,247,627	1,499.0

Table 13: Solutions found and computational cost of solving the small-sized instances with learning rate $\alpha = 0.3$ using CPLEX and CP Optimizer with warm start.

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
DAFJS01	[21,618, 28,107] 23.09%	2,625,310	8,412	3,600.0	[18,588, 23,460] 20.77%	11,970,328	3,600.0
DAFJS02	[23,418, 29,389] 20.32%	3,476,653	10,932	3,600.0	[18,533, 26,535] 30.16%	9,040,577	3,600.0
DAFJS03	[42,594, 53,688†] 20.66%	116,124	0	3,600.0	[41,271, 53,610] 23.02%	896,736	3,600.0
DAFJS04	[45,501, 54,082†] 15.87%	204,366	82	3,600.0	[44,062, 53,010] 16.88%	1,705,142	3,600.0
DAFJS05	[30,528, 45,862†] 33.44%	354,131	576	3,600.0	[23,378, 37,770] 38.1%	5,032,131	3,600.0
DAFJS06	[31,799, 51,925†] 38.76%	226,576	80	3,600.0	[23,364, 36,872] 36.63%	1,822,585	3,600.0
DAFJS07	[34,214, 57,193†] 40.18%	58,523	0	3,600.0	[33,601, 51,411] 34.64%	321,191	3,600.0
DAFJS08	[43,521, 62,159†] 29.98%	70,699	0	3,600.0	[43,331, 61,362] 29.38%	334,236	3,600.0
DAFJS09	[36,343, 48,680†] 25.34%	205,835	100	3,600.0	[22,460, 42,199] 46.78%	2,205,797	3,600.0
DAFJS10	[39,542, 58,695†] 32.63%	122,016	90	3,600.0	[23,440, 48,563] 51.73%	851,860	3,600.0
DAFJS11	[44,224, 67,594†] 34.57%	65,692	0	3,600.0	[44,061, 63,573] 30.69%	271,903	3,600.0
DAFJS12	[38,375, 68,322†] 43.83%	72,522	0	3,600.0	[35,262, 68,084] 48.21%	189,478	3,600.0
DAFJS13	[50,239, 61,794†] 18.7%	55,740	68	3,600.0	[20,981, 55,636] 62.29%	1,545,915	3,600.0
DAFJS14	[55,259, 76,862†] 28.11%	54,196	19	3,600.0	[23,996, 65,747] 63.5%	1,009,208	3,600.0
DAFJS15	[0, 72,288†] 100%	81,108	0	3,600.0	[33,814, 69,826] 51.57%	122,557	3,600.0
DAFJS16	[42,615, 76,204†] 44.08%	91,239	0	3,600.0	[42,465, 69,138] 38.58%	195,672	3,600.0
DAFJS17	[57,425, 73,219†] 21.57%	69,268	0	3,600.0	[20,260, 71,060] 71.49%	409,608	3,600.0
DAFJS18	[58,244, 78,862†] 26.14%	40,568	0	3,600.0	[21,838, 72,542] 69.9%	572,642	3,600.0
DAFJS19	[36,123, 66,412†] 45.61%	51,797	11	3,600.0	[35,267, 52,507] 32.83%	665,468	3,600.0
DAFJS20	[49,353, 78,778†] 37.35%	43,063	0	3,600.0	[29,175, 78,778†] 62.97%	198,407	3,600.0
DAFJS21	[52,221, 78,320†] 33.32%	62,815	0	3,600.0	[33,405, 75,364] 55.68%	231,834	3,600.0
DAFJS22	[44,636, 70,892†] 37.04%	65,715	0	3,600.0	[30,309, 66,797] 54.63%	192,435	3,600.0
DAFJS23	[31,584, 49,110] 35.69%	52,522	0	3,600.0	[31,047, 49,061] 36.72%	748,142	3,600.0
DAFJS24	[36,461, 57,411†] 36.49%	61,989	0	3,600.0	[32,194, 54,926] 41.39%	529,912	3,600.0
DAFJS25	[0, 82,055†] 100%	88,785	0	3,600.0	[38,155, 82,015] 53.48%	127,645	3,600.0
DAFJS26	[0, 81,480†] 100%	74,398	0	3,600.0	[36,963, 81,480†] 54.64%	107,805	3,600.0
DAFJS27	[0, 81,470†] 100%	80,755	0	3,600.0	[32,947, 81,470†] 59.56%	115,600	3,600.0
DAFJS28	[0, 62,568†] 100%	108,749	0	3,600.0	[36,392, 57,459] 36.66%	301,750	3,600.0
DAFJS29	[41,245, 72,938†] 43.45%	60,978	0	3,600.0	[40,983, 70,410] 41.79%	278,261	3,600.0
DAFJS30	[34,731, 61,147†] 43.2%	111,886	0	3,600.0	[31,515, 60,403] 47.83%	422,912	3,600.0
YFJS01	[57,818, 80,523] 28.2%	631,409	1,146	3,600.0	[53,549, 69,462] 22.91%	8,688,261	3,600.0
YFJS02	[67,561, 81,579†] 17.18%	1,651,737	2,124	3,600.0	[61,286, 72,465] 15.43%	20,961,495	3,600.0
YFJS03	32,538	705,691	4,291	163.1	32,538	5,882,775	240.8
YFJS04	35,883	5,680,590	20,543	2,326.3	35,883	35,490,989	2,482.2
YFJS05	[37,761, 44,693] 15.51%	5,265,253	14,605	3,600.0	[32,158, 41,034] 21.63%	37,821,835	3,600.0
YFJS06	[35,026, 49,730] 29.57%	1,268,993	3,064	3,600.0	[25,961, 42,164] 38.43%	21,994,678	3,600.0
YFJS07	[36,364, 50,566] 28.09%	3,446,288	9,359	3,600.0	[32,925, 41,394] 20.46%	28,140,293	3,600.0
YFJS08	32,573	5,126,300	8,163	1,451.9	32,573	921,923	139.7
YFJS09	[18,421, 28,354†] 35.03%	448,453	415	3,600.0	[15,000, 22,681] 33.87%	5,587,852	3,600.0
YFJS10	[32,897, 41,403] 20.54%	5,261,520	8,363	3,600.0	[28,226, 37,372] 24.47%	40,955,286	3,600.0
YFJS11	[44,029, 59,356†] 25.82%	753,893	930	3,600.0	[39,044, 47,813] 18.34%	10,343,776	3,600.0
YFJS12	[38,581, 69,052] 44.13%	707,117	1,038	3,600.0	[33,937, 49,132] 30.93%	9,351,799	3,600.0
YFJS13	[32,018, 48,686] 34.24%	910,842	1,168	3,600.0	[27,343, 38,681] 29.31%	12,959,067	3,600.0
YFJS14	[95,365, 129,428†] 26.32%	76,343	0	3,600.0	[94,991, 119,807] 20.71%	1,139,268	3,600.0
YFJS15	[90,045, 138,196†] 34.84%	84,512	0	3,600.0	[89,818, 123,190] 27.09%	1,334,191	3,600.0
YFJS16	[0, 127,055†] 100%	89,770	0	3,600.0	[86,618, 123,582] 29.91%	1,122,570	3,600.0
YFJS17	[0, 109,112†] 100%	76,340	0	3,600.0	[76,416, 109,112†] 29.97%	208,535	3,600.0
YFJS18	[0, 133,703†] 100%	90,335	0	3,600.0	[81,982, 133,611] 38.64%	197,872	3,600.0
YFJS19	[0, 107,055†] 100%	76,027	0	3,600.0	[62,129, 107,014] 41.94%	179,087	3,600.0
YFJS20	[0, 97,868†] 100%	60,504	0	3,600.0	[65,048, 96,534] 32.62%	198,912	3,600.0
Mean	[34173.86, 68097.18] 49.82%	825,318.70	1,911.58	3,462.83	[39009.64, 63142.2] 38.22%	5,717,964.02	3,441.25

Table 14: Solutions found and computational cost of solving the large-sized instances with learning rate $\alpha = 0.1$ using CPLEX and CP Optimizer with warm start.

to whether the exact methods are able to improve the solution provided by the constructive heuristics or whether the statistics improve only because the solvers return the solution they received as input. In the case of the MILP model instances, the initial solution is improved in

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
DAFJS01	[16,695, 26,067] 35.95%	1,867,058	5,324	3,600.0	[14,141, 21,861] 35.31%	12,852,459	3,600.0
DAFJS02	[20,341, 28,154†] 27.75%	577,583	1,370	3,600.0	[13,966, 24,746] 43.56%	18,869,445	3,600.0
DAFJS03	[31,052, 48,555†] 36.05%	84,156	0	3,600.0	[29,519, 46,422] 36.41%	1,344,012	3,600.0
DAFJS04	[33,568, 48,461†] 30.73%	183,960	44	3,600.0	[32,038, 45,677] 29.86%	1,406,727	3,600.0
DAFJS05	[25,167, 44,790†] 43.81%	324,450	373	3,600.0	[17,025, 33,314] 48.9%	3,620,699	3,600.0
DAFJS06	[25,478, 44,228†] 42.39%	162,369	137	3,600.0	[16,745, 33,357] 49.8%	2,409,876	3,600.0
DAFJS07	[23,877, 50,802†] 53%	61,634	0	3,600.0	[23,042, 45,742] 49.63%	398,145	3,600.0
DAFJS08	[30,551, 53,439†] 42.83%	47,840	0	3,600.0	[29,898, 51,973] 42.47%	625,148	3,600.0
DAFJS09	[28,845, 42,537†] 32.19%	130,367	100	3,600.0	[16,016, 36,539] 56.17%	1,722,093	3,600.0
DAFJS10	[30,021, 46,257†] 35.1%	63,949	10	3,600.0	[16,354, 40,354] 59.47%	1,779,988	3,600.0
DAFJS11	[29,853, 56,847†] 47.49%	55,377	0	3,600.0	[29,499, 55,646] 46.99%	205,807	3,600.0
DAFJS12	[26,886, 55,575†] 51.62%	99,798	0	3,600.0	[23,461, 55,575†] 57.78%	104,236	3,600.0
DAFJS13	[39,200, 51,317†] 23.61%	50,909	66	3,600.0	[14,433, 48,176] 70.04%	880,264	3,600.0
DAFJS14	[43,922, 62,724†] 29.98%	36,836	13	3,600.0	[16,518, 54,861] 69.89%	769,749	3,600.0
DAFJS15	[0, 54,353†] 100%	55,931	0	3,600.0	[22,324, 53,549] 58.31%	97,946	3,600.0
DAFJS16	[28,823, 65,550†] 56.03%	92,513	0	3,600.0	[28,167, 65,257] 56.84%	135,220	3,600.0
DAFJS17	[42,492, 63,177†] 32.74%	33,771	0	3,600.0	[13,683, 56,330] 75.71%	447,082	3,600.0
DAFJS18	[45,630, 61,831†] 26.2%	29,131	23	3,600.0	[14,813, 59,787] 75.22%	1,007,448	3,600.0
DAFJS19	[27,728, 58,001†] 52.19%	47,626	17	3,600.0	[24,250, 44,486] 45.49%	881,311	3,600.0
DAFJS20	[37,194, 63,588†] 41.51%	42,948	0	3,600.0	[19,568, 63,065] 68.97%	332,996	3,600.0
DAFJS21	[36,582, 63,202†] 42.12%	54,908	0	3,600.0	[22,142, 62,438] 64.54%	179,594	3,600.0
DAFJS22	[31,266, 56,115†] 44.28%	72,153	0	3,600.0	[19,800, 56,115†] 64.72%	146,289	3,600.0
DAFJS23	[23,845, 47,616†] 49.92%	40,481	0	3,600.0	[21,422, 42,266] 49.32%	488,429	3,600.0
DAFJS24	[26,527, 49,019†] 45.88%	56,184	0	3,600.0	[21,763, 48,525] 55.15%	346,533	3,600.0
DAFJS25	[0, 66,262†] 100%	111,844	0	3,600.0	[24,843, 66,200] 62.47%	88,055	3,600.0
DAFJS26	[0, 72,226†] 100%	77,334	0	3,600.0	[24,182, 71,874] 66.36%	101,029	3,600.0
DAFJS27	[0, 64,797†] 100%	64,933	0	3,600.0	[21,584, 64,781] 66.68%	145,287	3,600.0
DAFJS28	[25,167, 52,639†] 52.19%	58,308	0	3,600.0	[24,712, 50,995] 51.54%	193,331	3,600.0
DAFJS29	[29,463, 59,792†] 50.72%	66,255	0	3,600.0	[27,497, 57,222] 51.95%	246,809	3,600.0
DAFJS30	[27,084, 55,015†] 50.77%	91,519	0	3,600.0	[21,222, 54,467] 61.04%	306,321	3,600.0
YFJS01	[47,837, 84,152†] 43.15%	768,698	1,077	3,600.0	[39,902, 63,486] 37.15%	4,653,807	3,600.0
YFJS02	[59,193, 66,853†] 11.46%	1,587,076	2,455	3,600.0	[46,660, 66,853†] 30.21%	16,559,119	3,600.0
YFJS03	30,073	2,666,461	8,869	651.0	30,073	1,859,541	89.8
YFJS04	[29,362, 35,810] 18.01%	8,026,009	27,106	3,600.0	[21,857, 32,670] 33.1%	44,018,889	3,600.0
YFJS05	[34,400, 40,221] 14.47%	6,938,357	20,891	3,600.0	[24,921, 37,682] 33.86%	37,757,176	3,600.0
YFJS06	[31,101, 38,684] 19.6%	3,339,460	7,174	3,600.0	[19,541, 36,812] 46.92%	17,596,410	3,600.0
YFJS07	[31,513, 42,140] 25.22%	2,383,275	7,262	3,600.0	[26,063, 37,664] 30.8%	11,920,798	3,600.0
YFJS08	30,184	8,684,046	11,067	2,401.4	30,184	15,779,708	1,242.7
YFJS09	[14,785, 23,430] 36.9%	330,752	106	3,600.0	[11,324, 21,796] 48.05%	5,536,683	3,600.0
YFJS10	[30,116, 37,720] 20.16%	8,688,281	15,350	3,600.0	[22,264, 34,624] 35.7%	38,762,863	3,600.0
YFJS11	[35,760, 51,469†] 30.52%	653,855	652	3,600.0	[30,073, 44,294] 32.11%	16,060,438	3,600.0
YFJS12	[32,446, 59,898†] 45.83%	795,705	1,181	3,600.0	[26,939, 43,790] 38.48%	13,545,313	3,600.0
YFJS13	[25,469, 45,084†] 43.51%	466,368	549	3,600.0	[21,361, 35,266] 39.43%	8,512,041	3,600.0
YFJS14	[68,791, 109,469†] 37.16%	74,831	0	3,600.0	[68,423, 98,323] 30.41%	1,965,536	3,600.0
YFJS15	[65,453, 112,424†] 41.78%	78,095	0	3,600.0	[64,923, 105,088] 38.22%	5,880,443	3,600.0
YFJS16	[63,661, 106,855†] 40.42%	56,452	0	3,600.0	[62,865, 103,358] 39.18%	1,474,916	3,600.0
YFJS17	[0, 85,736†] 100%	84,708	0	3,600.0	[51,540, 85,611] 39.8%	277,616	3,600.0
YFJS18	[0, 99,429†] 100%	85,143	0	3,600.0	[55,101, 97,691] 43.6%	256,080	3,600.0
YFJS19	[0, 89,561†] 100%	109,068	0	3,600.0	[41,690, 89,561†] 53.45%	205,157	3,600.0
YFJS20	[0, 91,958†] 100%	111,180	0	3,600.0	[43,711, 91,958†] 52.47%	127,010	3,600.0
Mean	[28348.02, 57881.72] 51.02%	1,013,398.90	2,224.32	3,517.05	[27680.84, 53967.68] 48.71%	5,897,637.44	3,482.65

Table 15: Solutions found and computational cost of solving the large-sized instances with learning rate $\alpha = 0.2$ using CPLEX and CP Optimizer with warm start.

33 problems, while in the CP model instances the initial solution is improved in 134 problems. Without a warm start, in the instances where it is possible to find a provably optimal solution for both the MILP model and the CP model (5 instances), the cost of solving the CP models is 70.81% lower. In the case where provably optimal solutions are found in both cases using warm

instance	IBM ILOG CPLEX				IBM ILOG CP Optimizer		
	makespan	Effort measurement			makespan	Effort measurement	
		#iterations	#B&B Nodes	CPU		#branches	CPU
DAFJS01	[13,085, 22,616†] 42.14%	1,117,444	1,730	3,600.0	[10,730, 20,247] 47%	7,775,777	3,600.0
DAFJS02	[16,801, 26,183†] 35.83%	1,515,028	3,091	3,600.0	[10,500, 22,147] 52.59%	18,313,536	3,600.0
DAFJS03	[22,719, 43,964†] 48.32%	87,465	6	3,600.0	[21,152, 42,677] 50.44%	653,668	3,600.0
DAFJS04	[24,927, 43,281†] 42.41%	150,187	31	3,600.0	[23,278, 40,023] 41.84%	2,127,914	3,600.0
DAFJS05	[20,474, 38,237†] 46.46%	298,694	514	3,600.0	[12,401, 29,897] 58.52%	3,366,879	3,600.0
DAFJS06	[20,171, 38,007†] 46.93%	103,184	100	3,600.0	[12,004, 28,181] 57.4%	1,790,536	3,600.0
DAFJS07	[16,887, 41,285†] 59.1%	57,392	0	3,600.0	[15,801, 40,857] 61.33%	311,704	3,600.0
DAFJS08	[21,334, 46,323†] 53.95%	84,392	0	3,600.0	[20,629, 42,452] 51.41%	289,066	3,600.0
DAFJS09	[22,185, 41,418†] 46.44%	176,125	14	3,600.0	[11,423, 33,332] 65.73%	1,702,714	3,600.0
DAFJS10	[22,463, 38,771†] 42.06%	71,903	14	3,600.0	[11,408, 32,284] 64.66%	1,290,793	3,600.0
DAFJS11	[20,011, 47,805†] 58.14%	81,360	0	3,600.0	[19,709, 47,731] 58.71%	193,904	3,600.0
DAFJS12	[18,895, 45,735†] 58.69%	76,070	0	3,600.0	[15,608, 44,045] 64.56%	196,960	3,600.0
DAFJS13	[29,210, 42,319†] 30.98%	65,284	26	3,600.0	[9,929, 40,101] 75.24%	1,351,794	3,600.0
DAFJS14	[32,979, 49,459†] 33.32%	47,535	37	3,600.0	[11,318, 45,078] 74.89%	763,657	3,600.0
DAFJS15	[0, 49,206†] 100%	71,721	0	3,600.0	[14,671, 48,366] 69.67%	131,171	3,600.0
DAFJS16	[20,218, 56,727†] 64.36%	97,311	0	3,600.0	[18,661, 56,727†] 67.1%	171,678	3,600.0
DAFJS17	[29,862, 54,527†] 45.23%	29,116	0	3,600.0	[9,240, 48,773] 81.06%	499,025	3,600.0
DAFJS18	[32,271, 51,823†] 37.73%	29,115	12	3,600.0	[10,047, 47,845] 79%	670,899	3,600.0
DAFJS19	[21,074, 46,880†] 55.05%	44,072	9	3,600.0	[16,675, 43,407] 61.58%	654,923	3,600.0
DAFJS20	[25,682, 51,686†] 50.31%	58,048	0	3,600.0	[13,124, 51,061] 74.3%	333,744	3,600.0
DAFJS21	[0, 53,010†] 100%	66,397	0	3,600.0	[14,664, 50,104] 70.73%	234,319	3,600.0
DAFJS22	[21,312, 45,005†] 52.65%	85,113	0	3,600.0	[12,934, 45,005†] 71.26%	146,342	3,600.0
DAFJS23	[18,008, 39,249†] 54.12%	41,167	0	3,600.0	[14,782, 39,145] 62.24%	652,256	3,600.0
DAFJS24	[18,974, 44,851†] 57.7%	61,919	0	3,600.0	[14,688, 44,462] 66.97%	255,869	3,600.0
DAFJS25	[0, 51,964†] 100%	71,177	0	3,600.0	[16,174, 51,964†] 68.87%	83,334	3,600.0
DAFJS26	[0, 57,182†] 100%	103,993	0	3,600.0	[15,820, 57,182†] 72.33%	123,884	3,600.0
DAFJS27	[25,010, 58,138†] 56.98%	107,828	0	3,600.0	[14,138, 57,409] 75.37%	139,297	3,600.0
DAFJS28	[18,433, 42,898†] 57.03%	63,459	0	3,600.0	[16,781, 40,331] 58.39%	611,659	3,600.0
DAFJS29	[22,832, 51,606†] 55.76%	58,895	0	3,600.0	[18,452, 51,156] 63.93%	194,564	3,600.0
DAFJS30	[17,829, 43,859†] 59.35%	82,952	0	3,600.0	[14,275, 42,315] 66.26%	281,586	3,600.0
YFJS01	[39,168, 66,116] 40.76%	560,775	1,035	3,600.0	[29,808, 58,170] 48.76%	6,636,736	3,600.0
YFJS02	[48,544, 59,211] 18.02%	1,188,834	2,163	3,600.0	[35,526, 59,001] 39.79%	12,010,945	3,600.0
YFJS03	27,686	1,406,947	4,582	337.1	27,686	1,278,286	66.9
YFJS04	[25,715, 32,191] 20.12%	8,529,348	26,006	3,600.0	[18,161, 29,692] 38.84%	42,209,341	3,600.0
YFJS05	[31,090, 34,905] 10.93%	8,010,184	28,009	3,600.0	[19,314, 35,629] 45.79%	40,473,041	3,600.0
YFJS06	[26,650, 39,813] 33.06%	2,057,905	4,922	3,600.0	[14,714, 33,822] 56.5%	14,236,397	3,600.0
YFJS07	[27,041, 38,134] 29.09%	3,907,838	10,102	3,600.0	[20,397, 33,059] 38.3%	12,005,380	3,600.0
YFJS08	[27,020, 28,638] 5.65%	11,006,083	13,663	3,600.0	[17,787, 28,192] 36.91%	41,659,118	3,600.0
YFJS09	[12,166, 21,816] 44.23%	439,398	350	3,600.0	[9,525, 20,457] 53.44%	4,119,430	3,600.0
YFJS10	[28,406, 33,622] 15.51%	8,543,764	16,090	3,600.0	[17,808, 32,159] 44.63%	30,351,408	3,600.0
YFJS11	[30,906, 45,051†] 31.4%	770,066	1,119	3,600.0	[23,170, 41,095] 43.62%	7,467,430	3,600.0
YFJS12	[26,035, 48,493] 46.31%	381,810	116	3,600.0	[21,335, 39,039] 45.35%	7,124,912	3,600.0
YFJS13	[23,227, 40,119†] 42.1%	812,817	1,074	3,600.0	[16,668, 30,711] 45.73%	4,335,509	3,600.0
YFJS14	[49,960, 92,457†] 45.96%	83,456	0	3,600.0	[49,252, 91,042] 45.9%	1,297,229	3,600.0
YFJS15	[47,574, 96,547†] 50.72%	77,782	0	3,600.0	[46,947, 89,100] 47.31%	824,181	3,600.0
YFJS16	[47,654, 92,811†] 48.65%	55,873	0	3,600.0	[45,648, 87,080] 47.58%	1,706,868	3,600.0
YFJS17	[0, 73,682†] 100%	65,088	0	3,600.0	[34,763, 73,682†] 52.82%	110,226	3,600.0
YFJS18	[0, 87,059†] 100%	72,375	0	3,600.0	[37,042, 87,059†] 57.45%	111,311	3,600.0
YFJS19	[0, 74,431†] 100%	62,604	0	3,600.0	[27,976, 74,431†] 62.41%	138,828	3,600.0
YFJS20	[0, 72,481†] 100%	133,020	0	3,600.0	[29,376, 72,250] 59.34%	138,581	3,600.0
Mean	[21849.76, 49385.54] 51.47%	1,063,406.26	2,296.30	3,534.74	[19678.38, 46553.2] 57.73%	5,470,972.18	3,529.34

Table 16: Solutions found and computational cost of solving the large-sized instances with learning rate $\alpha = 0.3$ using CPLEX and CP Optimizer with warm start.

start (6 instances), the cost of solving the CP models is 41.86% lower. In short, it is challenging to find a proven optimal solution for large-sized instances, solving CP model instances costs less, CP models provide better quality feasible solutions when it is not possible to find a provably

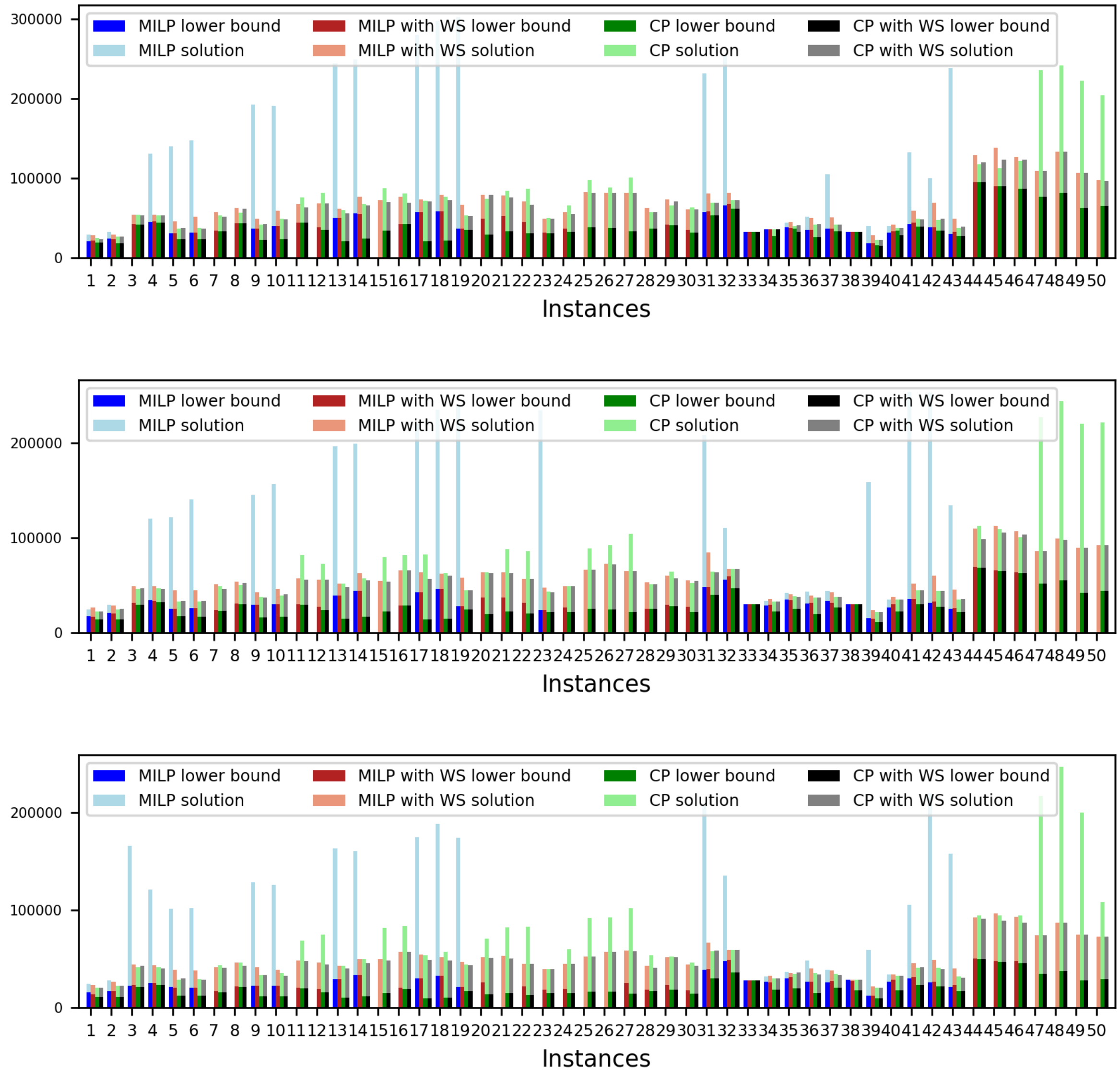


Figure 6: Graphical representation of the information contained in Tables 5–16. The figures at the top, middle and bottom correspond to α equal to 0.1, 0.2, and 0.3, respectively. Each figure shows, for each large-sized size instance, the solution found by solving the MILP and CP models with and without warm start, as well as the lower bounds found in these 4 cases. The numbers from 1 to 50 on the abscissa axis correspond to the 50 large-sized instances in the order presented in all the tables. That is, the instances from 1 to 30 correspond to the instances DAFJS01, …, DAFJS30 and the instances from 31 to 50 correspond to the instances YFJS01, …, YFJS20.

optimal solution, and solving MILP models provides better quality lower bounds. One way or another, whether using the MILP model or the CP model, with or without warm start, it was possible to find provably optimal solutions in only 7 large-sized instances and feasible solutions in all the remaining 143 large-sized instances.

6 Conclusions

In this work, we addressed the FJS scheduling problem with nonlinear routes and position-based learning effect. To the authors' knowledge, this combination, with potential application in a wide range of real-world industrial environments, has never been addressed in the literature. As a first step towards its efficient and effective solution, we introduced a set of 110 instances that transform into 330 instances by varying the learning rate $\alpha \in \{0.1, 0.2, 0.3\}$. By introducing MILP and CP models, an exact solver aided by constructive heuristics was able to provide 183 proven optimal solutions. Instances, their solutions, models and constructive heuristics are all available for download at `https://github.com/kennedy94/FJS`. We expect this benchmark test-set to guide the introduction and evaluation of new effective heuristic and metaheuristic approaches for the considered problem. In fact, this is the current line of research of the authors.

Declarations of interest: None.

Author contributions: All authors contributed equally in all phases of the development of this work.

Data availability: The datasets generated during and/or analysed during the current study are available in the GitHub repository, `https://github.com/kennedy94/FJS`.

References

[1] R. Alvarez-Valdes, A. Fuertes, J. M. Tamarit, G. Giménez, and R. Ramos. A heuristic to schedule flexible job-shop in a glass factory. *European Journal of Operational Research*, 165(2):525–534, 2005.

[2] J. L. Andrade-Pineda, D. Canca, P. L. Gonzalez-R, and M. Calle. Scheduling a dual-resource flexible job shop with makespan and due date-related criteria. *Annals of Operations Research*, 291(1):5–35, 2020.

[3] A. Azzouz, M. Ennigrou, and L. Ben Said. Scheduling problems under learning effects: classification and cartography. *International Journal of Production Research*, 56(4):1642–1661, 2017.

[4] E. G. Birgin, P. Feofiloff, C. G. Fernandes, E. L. de Melo, M. T. I. Oshiro, and D. P. Ronconi. A MILP model for an extended version of the flexible job shop problem. *Optimization Letters*, 8(4):1417–1431, 2014.

[5] E. G. Birgin, J. E. Ferreira, and D. P. Ronconi. List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility. *European Journal of Operational Research*, 247(2):421–440, 2015.

[6] D. Biskup. Single-machine scheduling with learning considerations. *European Journal of Operational Research*, 115(1):173–178, 1999.

[7] D. Biskup. A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research*, 188(2):315–329, 2008.

[8] Z.-C. Cao, C.-R. Lin, and M.-C. Zhou. A knowledge-based cuckoo search algorithm to schedule a flexible job shop with sequencing flexibility. *IEEE Transactions on Automation Science and Engineering*, 18(1):56–69, 2021.

[9] T. C. E. Cheng and G. Wang. Single machine scheduling with learning effect considerations. *Annals of Operations Research*, 98(1/4):273–290, 2000.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 4th edition, 2022.

[11] P. Y. Gan and K. S. Lee. Scheduling of flexible-sequenced process plans in a mould manufacturing shop. *The International Journal of Advanced Manufacturing Technology*, 20(3):214–222, 2002.

[12] J. Gao, M. Gen, and L. Sun. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing*, 17(4):493–507, 2006.

[13] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[14] J. N. D. Gupta and S. K. Gupta. Single facility scheduling with nonlinear processing times. *Computers and Industrial Engineering*, 14(4):387–393, 1988.

[15] A. Janiak, T. Krysiak, and R. Trela. Scheduling problems with learning and ageing effects: A survey. *Decision Making in Manufacturing and Services*, 5(1):19–36, 2011.

[16] G. A. Kasapidis, S. Dauzère-Pérès, D. C. Paraskevopoulos, P. P. Repoussis, and C. D. Tarantilis. On the multiresource flexible job-shop scheduling problem with arbitrary precedence graphs. *Production and Operations Management*, 32(7):2322–2330, 2023.

[17] G. A. Kasapidis, D. C. Paraskevopoulos, P. P. Repoussis, and C. D. Tarantilis. Flexible job shop scheduling problems with arbitrary precedence graphs. *Production and Operations Management*, 30(11):4044–4068, 2021.

[18] Y. K. Kim, K. Park, and J. Ko. A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. *Computers and Operations Research*, 30(8):1151–1171, 2003.

[19] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23(2):210–250, 2018.

[20] J. Y.-T. Leung, H. Li, and M. Pinedo. Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling*, 8(5):355–386, 2005.

[21] W. T. Lunardi, E. G. Birgin, P. Laborie, D. P. Ronconi, and H. Voos. Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. *Computers and Operations Research*, 123:105020, 2020.

[22] W. T. Lunardi, E. G. Birgin, D. P. Ronconi, and H. Voos. Metaheuristics for the online printing shop scheduling problem. *European Journal of Operational Research*, 293(2):419–441, 2021.

[23] G. Vilcot and J.-C. Billaut. A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. *European Journal of Operational Research*, 190(2):398–411, 2008.

[24] A. Vital-Soto, A. Azab, and M. F. Baki. Mathematical modeling and a hybridized bacterial foraging optimization algorithm for the flexible job-shop scheduling problem with sequencing flexibility. *Journal of Manufacturing Systems*, 54:74–93, 2020.

[25] H. M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[26] J. M. Wilson. Alternative formulations of a flow-shop scheduling problem. *Journal of the Operational Research Society*, 40(4):395–399, 1989.

[27] L. Yu, C. Zhu, J. Shi, and W. Zhang. An extended flexible job shop scheduling model for flight deck scheduling with priority, parallel operations, and sequence flexibility. *Scientific Programming*, 2017:1–15, 2017.